Universitat de Lleida
Escola Politècnica Superior
Enginyeria Tècnica en Informàtica de Sistemes

Treball de final de carrera

DESENVOLUPAMENT D'APLICACIONS WEB EN DJANGO SOBRE SERVIDOR APACHE, GESTIONADES AMB MÒDUL WSGI I RELACIONADES AMB BASE DE DADES POSTGRESQL

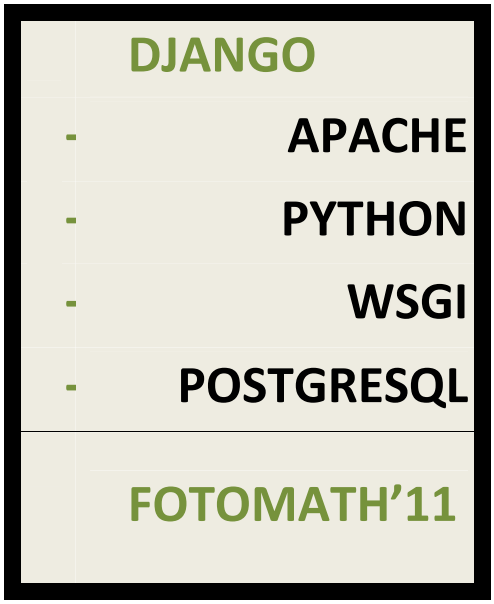
Autor: Carlos Bosch Gascón
Director: Josep Conde Colom
Setembre 2013

2013

TREBALL FINAL DE CARRERA

ENGINYERIA TÈCNICA EN INFORMÀTICA DE SISTEMES

CARLOS BOSCH GASCÓN



[FOTOMATH'11=DJANGO+APACHE+PYTHON+WSGI+POSTGRESQL]

[DESENVOLUPAMENT D'APLICACIONS WEB EN DJANGO SOBRE SERVIDOR APACHE, GESTIONADES AMB MÒDUL WSGI I RELACIONADES AMB BASE DE DADES POSTGRESQL]

Agraïments:

M'agradaria expressar el meu reconeixement i gratitud a totes aquelles persones que, gràcies a la seva col·laboració, han contribuït a la realització d'aquest projecte de fi de carrera:

En primer lloc, el meu més sincer agraïment a Josep Conde Colom, tutor d'aquest projecte, pel seu consell, ajuda i dedicació durant el desenvolupament d'aquest projecte.

Un agraïment especial a Neus Marsellés Fontanet per la seva ajuda i comprensió, i també agrair a la meva família el seu recolzament durant tot aquest període.

A tots vosaltres, moltes gràcies.

Índex de contingut

Índex de taules

Índex d'imatges

1.- INTRODUCCIÓ

El treball de final de carrera que presentem s'elabora per suplir les necessitats de gestió i organització del *II Concurs de Fotografia Matemàtica FotoMath 2011*.

Per fer-ho, s'ha tingut en compte el nombre de possibles participants, l'opinió dels organitzadors i la versió prèvia del sistema, que es detalla en les línies següents.

El concurs *FotoMath* va néixer amb l'objectiu principal de fer palesa la presència de les matemàtiques en la vida quotidiana, per tal de fer-les més properes al públic en general.

La primera edició va ser *FotoMath 2009* i es plantejava com un certamen de caràcter biennal on els participants al concurs farien arribar les seves fotografies i dades personals seguint unes bases concretes penjades a la web del concurs. Aquestes estipulaven, entre altres coses, que l'entrega del material s'havia de fer mitjançant correu electrònic, amb els diferents inconvenients que això comportava.

Els principals problemes que van sorgir durant la primera edició foren:

1. haver de controlar tots els correus electrònics que arribaven,
2. revisar que les imatges complissin els requisits per formar part del concurs i
3. que les dades dels concursants fossin correctes.

Aquests punts suposaven per a l'organització del concurs una enorme despesa de temps i esforç.

Així, per millorar la gestió dels participants al següent concurs, *FotoMath 2011*, ens vam reunir amb els seus creadors per veure quines necessitats tenien i poder trobar-hi una solució adaptada a les seves necessitats. Les conclusions extretes van ser:

1. es realitzaria un formulari web per a la introducció de les dades i l'enviament de les imatges,
2. es guardaria tota la informació entregada en una base de dades per a poder gestionar millor el material.

Amb aquestes premisses s'ha elaborat i estructurat el projecte de la següent manera:

- Una primera part amb la qual es pretén contextualitzar i introduir al lector en el món dels *Frameworks*, què són i com funcionen. A més, s'hi explicarà què és un patró *MVC (Model View Controller)*, el model d'aplicacions web *Django* i el llenguatge de programació *Python*.
- En una segona part, s'abordarà tot el que fa referència a:

  - la situació anterior,
  - la situació a la que s'ha arribat, la qual comprèn:
    - el maquinari amb el que s'ha treballat,
    - el programari que s'ha instal·lat i creat,
    - el funcionament de l'aplicació desenvolupada i la seva configuració,
  - l'entorn de l'aplicació.
- La tercera part del projecte s'ha reservat a l'estudi de la viabilitat del projecte, les millores per a futures edicions i les conclusions extretes de la feina feta.
- En els annexos, el lector pot trobar-hi tota la documentació i codi desenvolupat que no s'ha afegit en la segona part del projecte per tal de facilitar-ne la lectura.

2.- INTRODUCCIÓ ALS FRAMEWORKS

En aquest capítol s'explicarà que són els *Frameworks*, quins tipus n'existeixen i que hi tenen a veure els patrons de disseny MVC.

També s'explicarà que és Django, quines característiques principals té i quins requisits de sistema necessita.

2.1.- ELS FRAMEWORKS

En programació informàtica un *Framework* és considerat un entorn d'abstracció on el programari aporta funcionalitats genèriques al sistema les quals poden ser canviades segons la codificació de l'usuari, en aquest cas, el programador.

Aquestes funcionalitats, aquesta col·lecció de biblioteques de programari, proporcionen una *I*nterfície de *P*rogramació d'*A*plicacions coneguda com API.

Així, els entorns de treball disposen de característiques principals que les distingeix de les llibreries més típiques. Aquestes característiques són:

- **Inversió de control:** a diferència de les llibreries o les aplicacions d'usuari habituals, el programa general de flux de control no és regeix per qui fa la crida sinó pel *Framework*.

- **Comportament per defecte:** l'entorn de treball té un comportament per defecte.

- **Extensibilitat:** l'usuari pot ampliar les funcionalitats del *Framework* amb funcions específiques.

- **Codi no modificable:** tot i que l'usuari pot millorar o ampliar funcionalitats al seu gust hi ha part de l'entorn de treball que no podrà ser modificat ja que forma part del funcionament bàsic del propi *Framework*.

Així, la creació dels *Frameworks* s'origina per facilitar el desenvolupament de programari, permeten als dissenyadors i programadors dedicar més temps a conèixer les necessitats del sistema i no haver de dedicar-se a proveir el sistema de detalls de baix nivell, reduint així el temps de desenvolupament.

Una de les estratègies utilitzades habitualment en la creació de projectes de programari és la *Top-Down* que genera un resum del sistema sense especificar detalls. Els *Frameworks* segueixen la mateixa ideologia.

D'aquesta manera, el més interessant de cara al programador serà la definició de dades, estructures i processament. La seva manipulació i com diferents tipus d'usuaris necessiten introduir o extreure dades, quin tipus d'interfície utilitzaran i amb quins privilegis, formaran part del codi personalitzat pel programador.

En qualsevol cas, cada tipus d'operació, i cada codi d'interfície d'usuari serà escrit i integrat en l'entorn de treball fent les proves pertinents fins a la validació final.

D'acord amb el que hem vist fins al moment els entorns de treball de programari consten de "punts freds", que defineixen tota l'arquitectura del sistema, els components bàsics així com les relacions que hi ha entre ells; i "punts calents", que representen aquelles parts on el programador utilitza el seu propi codi i el del *Framework* per afegir funcionalitats específiques al seu projecte.

En la taula que segueix es mostren alguns dels usos dels *Frameworks*:

Framework o entorn de treball		
	De programari	Conjunt de llibreries o classes reutilitzables per a sistemes de programari.
	D'aplicacions	Entorn utilitzat per implementar estructures estàndard d'aplicacions per a sistemes operatius específics. Es van fer populars amb l'augment de les interfícies gràfiques d'usuari (GUI) ja que aquestes tendeixen a promoure una estructura estàndard per a les aplicacions.
	D'aplicacions web	Entorn de treball de programari per desenvolupar pàgines web dinàmiques, aplicacions web o serveis web.
	Suit ofimàtica	Suit d'aplicacions oficials per a DOS llançada en 1984 per córrer en el PC original d'IBM sota el sistema operatiu MS-DOS.
	De conceptes	Un conjunt de teories àmpliament acceptades, suficients per servir com els principis per a la investigació dins d'una disciplina particular.

Taula 1: Tipologia d'alguns dels entorns de treball existents

Per tant es pot dir que un framework és l'esquelet sobre el qual diversos objectes són integrats per donar diferents funcionalitats al projecte.

Un cop vist a grans trets la funcionalitat i els tipus de *Frameworks* ens centrarem en aquells que fan referència a la web i que podem trobar definits en la secció següent.

2.2.- FRAMEWORK WEB

En les línies que segueixen es farà un resum històric de l'aparició dels entorns de treball per a la web, es comentaran les característiques bàsiques d'aquests *Frameworks* i els motius pels quals ens són d'utilitat en aquest projecte.

Avui en dia, un entorn de treball d'aplicacions web és un entorn de treball de programari dissenyat per suportar el desenvolupament de les pàgines web dinàmiques, de les aplicacions web i de serveis web. Aquest entorn tindrà, doncs, la finalitat d'estalviar tasques comuns, com per exemple: proveir el sistema durant el desenvolupament web de llibreries per a bases de dades, plantilles o gestió de sessions.

En els seus principis, el disseny de la World Wide Web no era dinàmic. Es codificava HTML (*H*yper*T*ext *M*arkup *L*anguage) fet a mà i es publicava sobre servidors web. Qualsevol modificació de les pàgines publicades només podia ser feta pel seu autor.

Amb el temps es van introduir millores com l'estàndard CGI (*C*ommon *G*ateway *I*nterface) que proporcionava el que començaria a ser la interacció amb l'usuari i que permetia la connexió d'aplicacions externes amb els servidors web. L'inconvenient que tenia CGI era que podria afectar negativament la càrrega del servidor, ja que cada petició havia d'iniciar un procés per separat.

Els programadors, per la seva banda, volien una major integració amb el servidor web per tal de permetre més capacitat de trànsit a les aplicacions web.

Així, amb el servidor *Apache HTTP (H*ypertext *T*ransfer *P*rotocol*)*, per exemple, es suportaven els mòduls per a execucions de codi arbitrari o transmissió de peticions específiques de contingut dinàmic. Altres servidors web, com *Apache Tomcat*, es van dissenyar específicament per gestionar el contingut dinàmic en l'execució de codi escrit en llenguatge *Java*.

Gairebé al mateix temps, nous llenguatges es van desenvolupar específicament per al seu ús a la web, com ara *ColdFusion*, *PHP* i *A*ctive *S*erver *P*ages.

Mentre que la gran majoria de llenguatges disponibles per a programadors de pàgines web dinàmiques tenien llibreries per facilitar les tasques més generals, les aplicacions web sovint necessitaven llibreries específiques.

Amb el temps, apareixen els *Frameworks* web, on s'hi poden trobar llibreries útils per al desenvolupament web en un entorn únic i coherent que agilitza les tasques en aplicacions web. Exemples en són: *JavaEE, WebObjects, web2py, OpenACS, Ruby on Rails, Zend Framework* i *Django*, essent aquest últim l'escollit per al projecte.

La majoria dels entorns d'aplicacions web d'avui dia es basen en el patró de disseny MVC (*M*odel *V*iew *C*ontroller).

2.2.1.- PATRONS DE DISSENY I MVC

Quan és comença a programar ens dediquem a codificar petits trossos de codi. Amb la pràctica, és bastant normal que necessitem copiar el primer programa o trossos del mateix en un altre lloc i començar a modificar, ja que el codi que hem de fer s'assembla al que teníem, però no és exactament igual.

Quan un programador es veu obligat a fer aquest pas una vegada i una altra, a copiar i enganxar el mateix codi i modificar una i altra vegada, comença a pensar en la forma de fer aquest tros de codi de manera que la tasca de portar a un altre lloc sigui més còmoda per a ell. El programador comença a inventar diferents formes de fer aquest tros de codi de manera que sigui prou versàtil i configurable com per no haver-ho de retocar més. Hauria de ser suficient amb portar la llibreria ja compilada d'un programa a un altre.

En la història hi ha hagut milers de programadors amb aquest problema i milers d'ells han arribat a les mateixes solucions o formes de fer aquests trossos de codi de manera que siguin completament reutilitzables. Aquestes solucions són els "patrons de disseny" i en el seu moment "la banda dels quatre" (quatre gurus de la programació, *Erich Gamma, Richard Helm, Ralph Johnson i John Vlissides*) els van recopilar en un llibre: *Design Patterns. Elements of reusable Object-Oriented Software*.

Tot i que el llibre no va ser editat fins a la dècada dels 1990 la idea dels patrons de disseny va començar a finals dels '70.

Un patró de disseny, doncs, no és res més que una solució general a un problema.

Els patrons de disseny ens expliquen com fer el nostre codi seguint unes normes, de manera que el nostre codi sigui reutilitzable, es pugui canviar, ampliar la seva funcionalitat i un llarg etcètera sense necessitat de tocar aquest codi.

En la taula que segueix és mostren alguns dels patrons existents:

Patrons de disseny	**Creacionals**	Fàbrica abstracta	Mètode de fabricació	Constructor virtual
	Estructurals	Adaptador	Pont	Embolcall
	De comportament	Ordre	Intèrpret	Observador

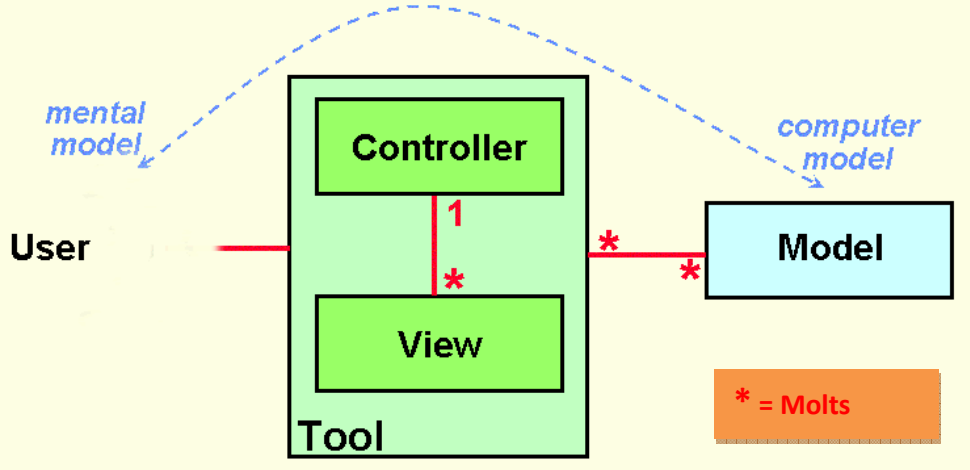
Taula 2: Relació dels principals patrons de disseny editats per la "Banda dels Quatre"

També és el cas de *MVC*. Aquest patró va ser ideat per *Trygve Reenskaug* el 1979 mentre treballava a *Xerox* amb *SmallTalk*.

Segons l'autor: "*He creat el patró Model-Vista-Controlador com una solució òbvia per al problema general de donar el control als usuaris sobre la seva pròpia informació des de múltiples perspectives*."

Per tant, i segons els comentaris de *Reenskaug* en la seva web, el Model Vista Controlador *"va ser creat per evitar la gran diferència entre el model mental de l'usuari humà i el model digital que existeix en l'ordinador. Aquesta solució MVC ideal suporta la idea de l'usuari en quan a vista i manipulació de la informació del domini de forma directa. L'estructura és útil quan l'usuari necessita veure els mateixos elements del model simultàniament en diferents contextos i/o des de diferents punts de vista."*

Una representació d'aquesta idea seria:



Imatge 1: Idea del MVC segons el seu autor *Trygve Reenskaug*

Aquest patró 'arquitectònic' separa el model de dades de la interfície d'usuari gràcies a unes regles de 'negoci'. La utilització d'aquet model es considera una bona pràctica, ja que codifica en mòduls i en promou la seva reutilització a més de permetre múltiples interfícies.

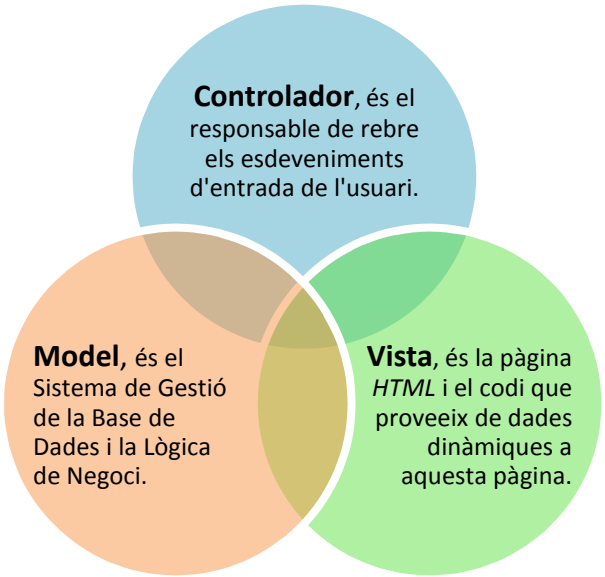
Encara que MVC està típicament associada amb els Frameworks, és essencialment una arquitectura. Això significa que es pot aplicar fins i tot sense un llenguatge orientat a objectes o una jerarquia de classes específiques.

Per exemple, amb només activar *jQuery()* i *bind()*, és possible la construcció de robustes aplicacions *MVC* en un cercador emprant *JavaScript*. La clau és simplement dividir les responsabilitats dels components *MVC* en seccions clarament definides de codi:

- Aquest codi que representa el model es fa càrrec de l'estat, la lògica de negoci, la persistència, i les notificacions.
  - La persistència es pot implementar a través de galetes o *AJAX*.
  - Les notificacions poden ser ateses per *jQuery.trigger()*.
- El codi que representa el punt de vista s'encarrega de consultar el model i la representació de la vista.
  - El codi de la vista pot ser implementat de moltes maneres, incloent la inserció de nodes *DOM* (*D*ocument *O*bject *M*odel) *HTML* o *CSS* (*C*ascading *S*tyle *S*heets).

El codi que representa el controlador s'encarrega de la inicialització del model i el cablejat dels esdeveniments entre els elements *HTML* de la vista i el controlador de *DOM* i entre el model i el codi de la vista, utilitzant *jQuery.bind()*.

Així, el patró *MVC* és podria entendre com:

- 

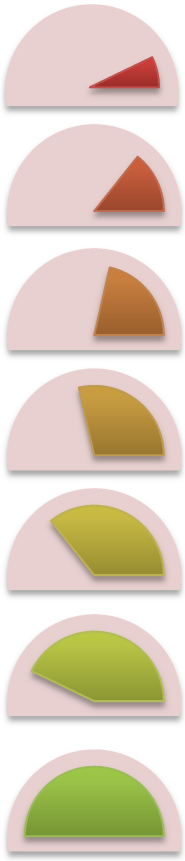
La finalitat d'aquest patró de disseny és millorar la reutilització mitjançant el desacoblament entre la vista i el model.

Els elements del patró són els següents:

Model	És responsable d'accedir a la capa d'emmagatzematge de dades. L'estructura ideal és que el model sigui independent del sistema d'emmagatzematge
	És qui defineix les regles de negoci (la funcionalitat del sistema). Un exemple en seria: *"Si la mercaderia demanada no es troba en el magatzem, consultar el temps de lliurament estàndard del proveïdor"*.
	També porta un registre de les vistes i controladors del sistema.
	Si estem davant d'un model actiu, notificarà a les vistes els canvis que en les dades pugui produir un agent extern. Per exemple, un fitxer que actualitza les dades, un temporitzador que desencadena una inserció, etcètera.
Controlador	Rep els esdeveniments d'entrada (un clic, un canvi en un camp de text, etc.).
	Conté regles de gestió d'esdeveniments, del tipus *"SI Esdeveniment Z, llavors Acció W"*. Aquestes accions poden suposar peticions al model o bé a les vistes. Una d'aquestes peticions a les vistes pot ser una crida al mètode *"Actualitzar ()"*. Una petició al model pot ser *"Coneixer_temps_entrega(nova_ordre_venta)"*.
Vista	És responsable de rebre dades del model i els mostra a l'usuari
	Tenen un registre del seu controlador associat.
	Poden donar el servei d' "Actualització ()", perquè sigui invocat pel controlador o pel model (quan és un model actiu que informa dels canvis en les dades produïdes per altres agents).

Taula 3: Relació de responsabilitats dins el patró MVC

Encara que es poden trobar diferents implementacions de MVC el flux general entre el sistema i l'usuari humà podria ser com el que segueix:



L'usuari interactua amb la interfície d'usuari d'alguna manera (per exemple, l'usuari prem un botó, enllaç, ...).

El **controlador rep** (per part dels objectes de la interfície-vista) la **notificació de l'acció** demanada per l'usuari.

El **controlador gestiona l'esdeveniment** que arriba, freqüentment a través d'un gestor d'esdeveniments (handler) o callback.

El **controlador accedeix al model**, actualitzant-lo, possiblement modificant de forma adequada a l'acció demanada per l'usuari (per exemple, el controlador actualitza el carro de la compra de l'usuari).

Els controladors complexos estan sovint estructurats usant un patró que **encapsula les accions** i simplifica la seva extensió.

El **controlador delega** als objectes de la vista la tasca de desplegar la interfície d'usuari.

La **vista obté les seves dades del model** per generar la interfície apropiada per a l'usuari on es reflecteix els canvis en el model (per exemple, produeix un llistat del contingut del carro de la compra).

Imatge 2: Flux de control de MVC

La majoria dels entorns *MVC* segueixen una arquitectura *"Push"*. Aquests entorns utilitzen les accions que realitzen els tractaments requerits i després "empenyen" les dades cap a la vista de resultats. En són exemples *Ruby on Rails* i *Django*.

Una alternativa a aquesta arquitectura és *"Pull"*. En aquest cas és comença amb la capa de visualització on s'hi mostren els resultats de tants controladors com es necessita per, en molts casos, una sola vista.

Per tant, les avantatges que és poden tenir en compte a l'hora de treballar amb *MVC* són:

- És possible **tenir diferents vistes** per a un mateix model (representació d'un conjunt de dades com una taula o llista d'imatges).
- És possible **construir noves vistes** sense necessitat de modificar el model subjacent.
- Proporciona un **mecanisme de configuració** a components complexos molts més tractable que el purament basat en esdeveniments (el model es pot veure com una representació estructurada de l'estat de la interacció).

2.2.2.- FRAMEWORK WEB

Així doncs un cop definit el patró *MVC* retornem a concretar els *Frameworks web*, les característiques dels quals és detallen en els següents punts:

Plantilles web	Les pàgines web dinàmiques solen tenir dues parts: Una serà *HTML estàtic*; l'altra part estarà formada per codi que generarà aquest HTML. Si considerem l'exemple d'un comerciant de fruita amb 25 varietats per vendre, en una pàgina web estàtica aquest comerciant hauria de crear 25 pàgines, una per cada un dels seus productes amb les diferents característiques de cada un; En una web dinàmica aquest comerciant només haurà de connectar la pàgina web dinàmica a la base de dades on hi consten les característiques de les 25 varietats. En una plantilla, les variables del llenguatge de programació és poden inserir sense utilitzar codi, d'aquesta manera és perd la necessitat de conèixer el llenguatge a l'hora de fer modificacions en la web. Així és crea una sintaxi que diferencia l'HTML de les variables.
Base de dades	Molts entorns d'aplicacions Web creen un sistema unificat [API] per a un motor de base de dades. D'aquesta manera les aplicacions web poden treballar amb una varietat de bases de dades sense canvis en el codi, i permetent als programadors treballar amb conceptes d'alt nivell.

URL	L'assignació d'URL de l'entorn de treball facilita el mecanisme pel qual el framework interpreta les URL. Alguns entorns com Django coincideix l'URL donada, no la modifica, en contra dels patrons predeterminats que utilitzen expressions regulars. Altres entorns tradueixen l'URL donada en una que el motor reconeix. Un sistema d'assignació d'adreces URL que utilitza el reconeixement de patrons o la reescriptura d'URL permet adreces URL més "amigables", augmentant la simplicitat del lloc i permetent una millor indexació pels motors de cerca. Per exemple, una URL que acaba amb "/ pag.cgi?cate=entrants&topic=amanida" es pot canviar a simplement "/pag/entrants/amanida". Això fa que l'URL sigui més fàcil de llegir i proporciona als motors de cerca millor informació sobre l'arquitectura del lloc.
Configuració automàtica	Alguns entorns redueixen la configuració de les aplicacions web mitjançant l'ús de la introspecció i/o després de seguir certes convencions. Per exemple, molts entorns de Java utilitzen Hibernate com una capa de persistència, la qual cosa pot generar un esquema de base de dades en temps d'execució capaç de mantenir la informació necessària. Això permet que el dissenyador de l'aplicació pugui dissenyar objectes de negoci sense necessitat de definir explícitament un esquema de base de dades. En Ruby on Rails també pot funcionar en sentit invers, és a dir, definir les propietats dels objectes del model en temps d'execució basant-lo en un esquema de base de dades.
Seguretat	Alguns entorns de treball d'aplicacions web disposen d'entorns amb autenticació i autorització de manera que permeten al servidor web identificar els usuaris de l'aplicació, i restringir l'accés a les funcions sobre la base d'alguns criteris definits.
Memòria cau	Servirà per reduir l'ús d'ample de banda, la càrrega del servidor entre altres utilitats. D'aquesta manera la memòria cau web emmagatzema còpies dels documents que passen a través d'ella. D'aquesta manera les sol·licituds posteriors poden ser ateses des de la mateixa memòria cau.

Un cop vistes les característiques ens centrarem en el *Framework* utilitzat per a aquest projecte.

2.3.- DJANGO

Va ser desenvolupat per gestionar diverses pàgines orientades a notícies de la World Company de Lawrence, Kansas, i va ser alliberada al públic sota una llicència *BSD*[1] el juliol de 2005.

Django és avui un *Framework* de desenvolupament web de codi obert, escrit en *Python*, que compleix en certa mesura el paradigma del *M*odel *V*ista *C*ontrolador.

Python és usat en totes les parts del *Framework*, fins i tot en configuracions, arxius, i en els models de dades.

L'objectiu fonamental de *Django* és facilitar la creació de llocs web complexos posant èmfasi en la reutilització, la connectivitat i extensibilitat de components, el desenvolupament ràpid i el principi *"No et repeteixis"* (*DRY*, de l'anglès *Your say Repeat Yourself*).

Django també proporciona una opció administrativa *CRUA* (crear, llegir, actualitzar i eliminar) de la interfície que es genera de forma dinàmica a través de la introspecció i configurar a través de models d'administració.

Algunes pàgines utilitzen *Django* des de fa temps, com la web d'*Spotify*. Aquesta és una pàgina des de la qual es permet escoltar música a traves d'Internet sense haver de comprar-la o descarregar-la. A més, el seu aplicatiu permet gestionar les biblioteques musicals de l'usuari des de qualsevol dispositiu digital.

El seu nom li ve donat en honor al guitarrista de jazz *Django Reinhardt*.

2.3.1.- CARACTERÍSTIQUES GENERALS DE DJANGO

Els orígens de *Django* com administrador de pàgines de notícies són evidents en el seu disseny, ja que proporciona una sèrie de característiques que faciliten el desenvolupament ràpid de pàgines orientades a continguts.

Per exemple, en lloc de necessitar que els desenvolupadors escriguin controladors i vistes per a les àrees d'administració d'una pàgina, *Django* incorpora una aplicació per administrar els continguts.

[1] *Llicència BSD: La Llicència de Distribució de Programari de Berkeley (anglès: Berkeley Software Distribution o BSD) no imposa cap restricció als desenvolupadors de programari pel que fa a la utilització posterior del codi en derivats i llicències d'aquests programes. Aquest tipus de llicència permet als programadors utilitzar, modificar i distribuir a tercers el codi font i el codi binari del programari original amb o sense modificacions. Els treballs derivats poden optar a llicències de codi obert o comercial.*

Aquesta aplicació es pot incloure com a part de qualsevol pàgina feta amb *Django* i pot administrar diverses pàgines fetes amb *Django* a partir d'una sola instal·lació. Permet la creació, actualització i eliminació d'objectes de contingut, portant un registre de totes les accions realitzades sobre cada un.

Django també proporciona una interfície per administrar els usuaris i els grups d'usuaris (incloent una assignació detallada de permisos).

A més dels arxius que s'han comentat *Django* també inclou:

- Un sistema de redirecció d'*URL*.
- Un sistema de comentaris.
- Eines per a sindicar contingut via *RSS* i/o *Atom*.
- Un servidor web independent i lleuger per a desenvolupament i proves.
- Un sistema de validació que permet diferenciar entre les etiquetes *HTML* i els valors susceptibles de ser guardats en la base de dades.
- Un entorn d'emmagatzematge de memòria cau que pot ser utilitzat per qualsevol dels mètodes de memòria cau.
- Suport per les classes de *Middleware* que poden intervenir en diverses etapes de processament de sol·licituds i dur a terme les funcions habituals.
- Un sistema intern que permet comunicar esdeveniments entre components d'una aplicació a través de senyals preestablertes.
- Un sistema d'internacionalització, incloent les traduccions dels propis components de *Django* en diferents idiomes.
- Un sistema de serialització que pot produir i llegir representacions *XML(*e*X*tensible *M*arkup *L*anguage*)* i/o *JSON (J*avaScript *O*bject *N*otation*)* de les instàncies de model de *Django*.
- Un sistema per estendre les capacitats del motor de plantilles.
- Una interfície creada amb *Python* per a unitàries.

El nucli de *Django* consisteix en una assignació entre objecte i relació que hi ha entre:

- els models de dades (definit com les classes de *Python*) i una base de dades relacional ("*model.py*"),
- un sistema per processar les sol·licituds amb un sistema de plantilles web ("*views.py*") i una expressió regular obtinguda per *URL* ("*Controller*").

Tot i això existeixen diferències entre la nomenclatura *Django* i el patró *MVC*.

Aquest fet es deu a que el disseny de *Django* no es va voler vincular a res en particular sinó que es volia desenvolupar una eina que funcionés el millor possible.

Si bé és cert que s'assembla molt a la implementació del patró *MVC*, per *Django* la *view* descriu **quines** dades seran presentades i no **com** es veuran. En el format de les dades és on entren en joc els *templates*.

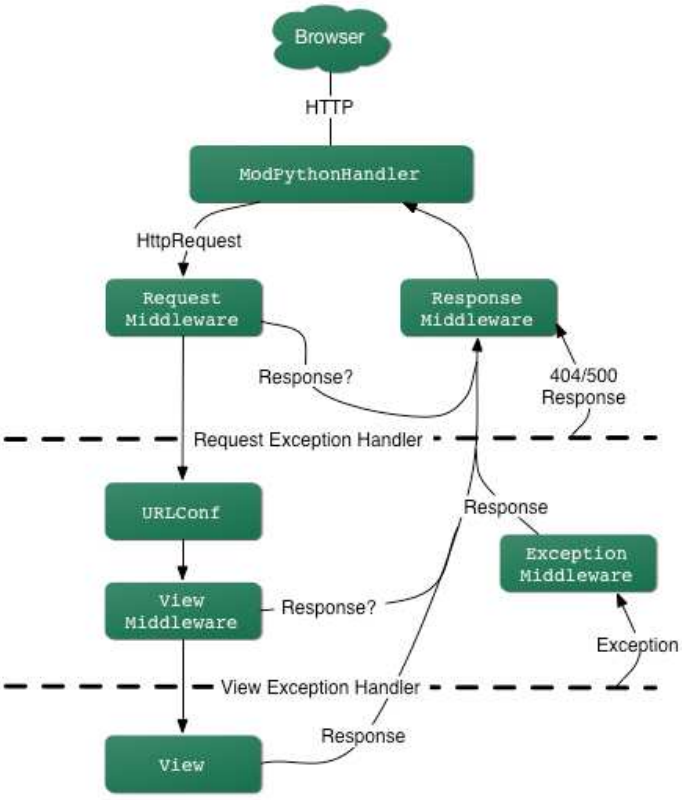
Django aparenta implementar el patró *MVC*, però el "*controlador*" de *MVC* és anomenat "*vista*" en *Django* i la "*vista*" de *MVC* és "*template*" en *Django*.

Es diu que el *controller* d'un *MVC* clàssic està representat pel propi *Framework*. És a dir, el sistema que envia un *request* a la *vista* corresponent, d'acord amb la configuració d'*URL* de *Django* (arxiu de configuració).

En el cas de voler fer una correspondència clara entre *Django* i *MVC* seria un *Framework MTV*: *M*odel, *T*emplate, *View*.

Tenint en compte l'arquitectura, veurem a grans trets com es processa un *request* en *Django*.

El flux de dades es pot representar com:



Imatge 3: Flux complet de *Django* en petició i resposta

Quan *Django* rep un petició *request*, el primer que es fa és crear un objecte *HTTPrequest* que la representa i servirà com a abstracció per treballar sobre diferents servidors.

Després es realitza la resolució de la *URL*. Això consisteix en seleccionar la funció del *view* (a partir de la *URL* especificada en el *request*) que participarà en la creació del *response*.

Una vegada que hem resolt quina funció resoldrà la *URL* especificada, s'invoca a la funció *view* amb la *petició* com a primer paràmetre. Aquí es realitza el treball pesat com:

- consultes a la base de dades,
- càrrega de *templates* i generació d'*HTML*.

Es retorna un objecte *HTTPresponse* o una excepció.

A més, *Django* proveeix tres punts diferents en els que permet executar classes *Middleware*[2], prèviament definides a l'arxiu de configuració. Una mateixa classe pot executar-se en més d'un punt, aquestes són les opcions:

- ***Request middleware*:** s'executa després de crear l'objecte sol·licitud *HTTP*, però abans de resoldre la *URL*, permetent modificar l'objecte *petició* o tornar una *resposta* pròpia abans que la resta de l'aplicació s'executi.

- ***View middleware***: és executat després de la resolució de la *URL*, però abans d'executar la *vista* corresponent. Permet executar operacions abans i després de l'execució de la vista. La *vista* podria arribar a no executar-se en absolut.

- ***Response middleware***: s'executa al final, després que l'objecte *response* hagi estat creat i abans de lliurar-lo al client. Es utilitzat per realitzar les modificacions finals.

2.3.2.- REQUISITS MÍNIMS DE SISTEMA

Django per treballar necessita tenir instal·lat:

- Python
- Servidor web i mod_wsgi (*W*eb *S*erver *G*ateway *I*nterface)
- Motor de base de dades. Segons quin s'instal·li és necessari afegir més paquets.

[2] *Middleware és un programari que assisteix a una aplicació per interactuar o comunicar-se amb altres aplicacions, software, xarxes, maquinari i / o sistemes operatius. Aquest simplifica el treball dels programadors en la complexa tasca de generar les connexions que són necessàries en els sistemes distribuïts. D'aquesta manera es proveeix una solució que millora la qualitat de servei, seguretat, enviament de missatges, directori de servei, etc*

3.- FOTOMATH

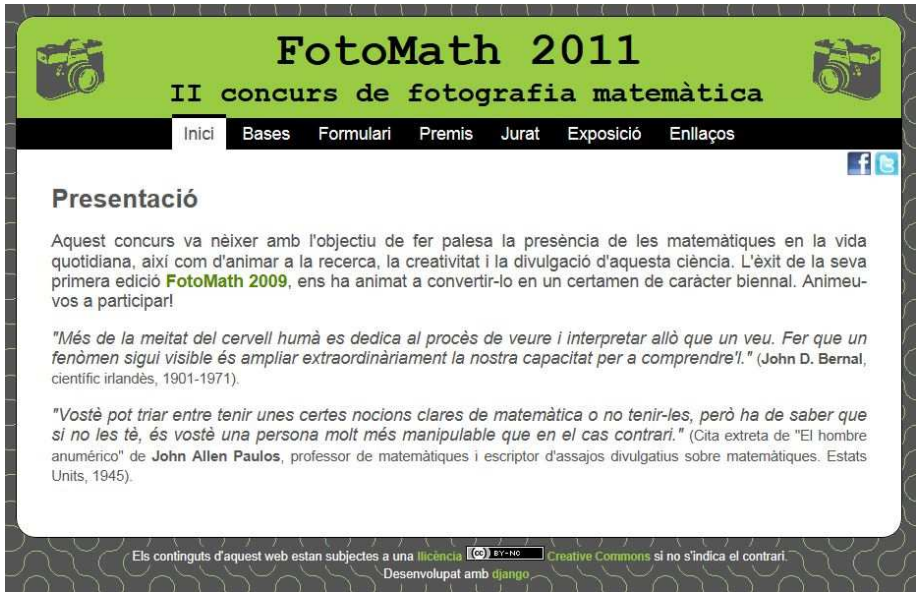
En aquest capítol es parlarà sobre la web de *FotoMath* en la versió anterior i quines eren les necessitats que s'havien d'implementar en el nou projecte, tant en el servidor de proves com en el servidor final.

També s'explicarà el programari instal·lat finalment, què fa i com funciona. Es detallarà el procés d'instal·lació i configuració del mateix juntament amb els motius que han dut a fer-lo d'aquesta manera.

3.1.- WEB FOTOMATH 2011

El projecte està centrat en desenvolupar dues aplicacions. Per un costat, es gestionaran els participants al concurs. Per l'altre, es gestionaran les imatges que aquests hi faran arribar. Això vol dir que aquestes dues noves aplicacions s'hauran d'adaptar a l'entorn ja desenvolupat per al primer concurs.

L'estructura de FotoMath[3] és una pàgina web on es pot observar el nom del certamen en la part superior i un menú just a sota amb les diferents seccions que disposa la web.



Imatge 4: Web FotoMath

[3] *http://www.fotomath.udl.cat*

El menú consta de set botons, amb les següents opcions:

1. Inici (Presentació de la web)
2. Bases (Bases al concurs)
3. Formulari (Formulari a emplenar per participar en el concurs)
4. Premis (Els premis per als guanyadors del concurs)
5. Jurat (Fotografia i descripció dels components del jurat)
6. Exposició (Apartat on es representen totes les imatges del concurs)
7. Enllaços (Diferents enllaços a altres concursos i galeries matemàtiques)

3.2.- ESPECIFICACIONS DEL SERVIDOR FOTOMATH

El servidor en el qual es troba allotjada la web de *FotoMath*, és una màquina virtual de la pròpia UdL, amb sistema operatiu *CentOS* 5 de Linux.

Dins d'aquest servidor virtual s'hi ha instal·lat:

- *Apache* versió 2.2.3
- Mòdul *WSGI* versió 3.3
- *Python* versió 2.7.1
- *Django* versió 1.2.5
- *Psycopg2* versió 2.4
- *PostgreSQL* versió 9.0

3.3.- ESPECIFICACIONS DEL SERVIDOR DE PROVES

El servidor de proves és el meu ordinador personal. En el que s'ha instal·lat el mateix programari que en el servidor *FotoMath* per prevenir possibles problemes d'incompatibilitats a l'hora de desenvolupar el programari.

Aquest equip de proves s'ha muntat amb un sistema operatiu *Ubuntu* 11.4 ja que aquest era el que s'adaptava millor a les nostres necessitats en aportar de sèrie *Python* 2.7.1.

Pel que fa a la resta de programari instal·lat no hi ha variació respecte el servidor *FotoMath*.

En els apartats que segueixen s'ampliaran les característiques del programari descrit. D'aquesta manera el lector coneixerà millor els elements instal·lats en el servidor i en la màquina de proves.

3.4.- DESCRIPCIÓ DEL PROGRAMARI INSTAL·LAT

3.4.1.- PYTHON

Python és un llenguatge de programació interpretat d'alt nivell i propòsit general d'on en destaca la llegibilitat del codi. Així, és disposa d'una sintaxi molt clara, amb una biblioteca estàndard àmplia.

Un tret característic de *Python*, que el fa únic entre els llenguatges de programació, és l'ús de la sagnia de delimitadors de bloc, que millora més encara la llegibilitat. A més, suporta múltiples paradigmes de programació. No és limita a la programació orientada a objectes sinó que també pot treballar amb els estils de programació funcional. Una altra característica és que és "*dynamically typed*" fent que la comprovació de la tipificació es faci durant l'execució i no la compilació. Així una variable pot fer referència a un valor de qualsevol tipus.

La implementació de referència de *Python* (*CPython*) és gratuït, és programari de codi obert, té un model de desenvolupament basat en la comunitat tot i que és administrat per la fundació sense ànim de lucre *Python Software Foundation*.

Aquest llenguatge de programació s'utilitza freqüentment per a aplicacions web, per exemple, a través de *mod_wsgi* (*W*eb *S*erver *G*ateway *I*nterface) per al servidor web *Apache*, essent una API estàndard per facilitar aquestes aplicacions.

Python també s'utilitza en *Frameworks* d'aplicacions web com *Django*, *Pylons*, *TurboGears*, *web2py*.

Les biblioteques com *NumPy*, *SciPy* i *Matplotlib* permeten utilitzar *Python* en la computació científica de manera efectiva.

Python ha estat incorporat amb èxit en una sèrie de productes de programari com un llenguatge de script, en el programari de mètode dels elements finits, com *ABAQUS*, paquets d'animació 3D, com *Houdini*, *Maya*, *MotionBuilder*, *Softimage*, *Cinema 4D*, *BodyPaint 3D* i programes d'imatges 2D com *GIMP*, *Inkscape*, *Scribus* i *Paint Shop Pro*.

Fins i tot s'ha utilitzat en diversos videojocs i ha estat adoptat com a primer dels tres llenguatges de programació disponibles a *Google App Engine* (des del maig de 2011), desbancant *Java* i *Go*.

A causa de les seves similituds amb *Lisp*, *Python* també s'ha utilitzat en intel·ligència artificial.

Per a molts sistemes operatius, *Python* és un component estàndard i ve amb la majoria de distribucions *Linux*, *NetBSD*, *OpenBSD* i *Mac OS X* podent-se utilitzar des de la terminal. Una sèrie de distribucions de *Linux* utilitzen instal·ladors escrits en *Python*: *Ubuntu* utilitza l'instal·lador *Ubiquity*, mentre que *Red Hat Linux* i *Fedora* utilitzen l'instal·lador *Anaconda*. *Gentoo Linux* fa un ús *Python* en el seu sistema de gestió de paquets. *Pardus* l'utilitza per a l'administració i durant l'arrencada del sistema.

Entre els usuaris de *Python* podem trobar *YouTube* o el client *Bit Torrent* original. Les grans empreses que fan ús de *Python* són *Google*, *Yahoo*, *CERN*, *NASA*, *ILM*[4] i *ITA*[5].

3.4.2.- APACHE2

L'Apache HTTP Server, conegut com *Apache*, és un programari de servidor web que destaca per jugar un paper clau en el creixement inicial de la *World Wide Web*.

En 2009 es va convertir en el primer programari de servidor web a superar la fita de 100 milions de llocs web.

Apache va ser la primera alternativa viable al servidor web *Netscape Communications Corporation* (actualment conegut com a servidor web d'*Oracle iPlanet*), i des d'aleshores ha evolucionat per competir amb altres servidors web en termes de funcionalitat i rendiment.

Normalment s'executa en un sistema tipus *Unix* encara que està disponible per a una àmplia varietat de sistemes operatius, incloent *GNU*, *FreeBSD*, *Linux*, *Solaris*, *NetWare* de *Novell*, *AmigaOS*, *Mac OS X*, *Microsoft Windows*, *OS/2*, *TPF* i *eComStation*.

Aquest servidor web suporta diverses funcionalitats, moltes implementades com a mòduls compilats que estenen la funcionalitat del nucli. Aquests poden incloure des de suport a llenguatges de programació de servidor fins a esquemes d'autenticació.

Algunes interfícies de llenguatge comunes suporten *mod_perl*, *mod_python*, *mod_wsgi*, *Tcl*, i *PHP*. Alguns mòduls d'autenticació populars inclouen *mod_access*, *mod_auth* i *mod_digest*.

A més, *Apache* ofereix missatges d'error personalitzables, bases de dades d'autenticació basades en *SGBD (S*istema de *G*estió de *B*ases de *D*ades*)*, i negociació de continguts. També està suportat per diverses interfícies gràfiques d'usuari (*GUIs*) que permeten configurar el servidor més fàcil i intuïtivament.

[4] *Industrial Light & Magic: Companyia d'efectes visuals fundada per George Lucas i guanyadora d'un Oscar.*
[5] *ITA Software: companyia fundada per científics del MIT l'any 1996 i adquirida per Google en 2010.*

L'arquitectura del servidor *Apache* és per tant molt modular. El servidor consta d'una secció *core* i molta de la funcionalitat que es podria considerar bàsica per un servidor web està en forma de mòduls. Alguns d'aquests són:

- ***mod_ssl*** - Comunicacions segures via TLS.
- ***mod_rewrite*** - Reescriptura de direccions servides (se sol utilitzar per transformar pàgines dinàmiques com .php a pàgines estàtiques .html).
- ***mod_dav*** - Suport del protocol WebDAV (RFC 2518).
- ***mod_deflate*** - Compressió transparent amb l'algoritme *deflate* del contingut enviat al client.
- ***mod_auth_ldap*** - Permet autenticar usuaris a un servidor *LDAP*.
- ***mod_proxy_ajp*** - Connector per enllaçar amb el servidor *Jakarta Tomcat* de pàgines dinàmiques *Java* (*servlets* i *JSP*).

El servidor de base es pot estendre mitjançant la inclusió de mòduls externs, entre els quals trobem:

- ***mod_perl*** - Pàgines dinàmiques amb Perl.
- ***mod_php*** - Pàgines dinàmiques amb PHP.
- ***mod_python*** - Pàgines dinàmiques amb Python.
- ***mod_rexx*** - Pàgines dinàmiques amb REXX i Object REXX.
- ***mod_ruby*** - Pàgines dinàmiques amb Ruby.
- ***mod_aspdotnet*** - Pàgines dinàmiques amb .NET_de_Microsoft.

L'hostatge virtual permet a una instal·lació d'*Apache* servir molts llocs web diferents. Per exemple, una màquina, amb una instal·lació pot servir al mateix temps www.exemple.com, www.prova.com, exemple.prova.com, etc.

Apache és desenvolupat i mantingut per una comunitat oberta de desenvolupadors sota *Apache Software Foundation*. A més, es troba publicat sota la seva pròpia llicència i és programari de codi obert.

Des d'abril de 1996, és considera el programari de servidor *HTTP* més popular. El maig de 2011 es va estimar que s'utilitzava en el 63% de tots els llocs web i el 66% dels més actius.

3.4.3.- MÒDUL WSGI

Mod_wsgi és un mòdul d'*Apache HTTP Server* creat per *Graham Dumpleton* que proporciona una interfície compatible *WSGI* per allotjar les aplicacions web basades en *Python* 2.3 o superior que treballen sobre *Apache*. Des de la versió 3.0, *mod_wsgi* suporta *Python* 2.6 i 3.1.

Per tant *mod_wsgi* és una alternativa a *mod_python*, *CGI* i *FastCGI* per les solucions d'integració web en *Python*.

3.4.4.- PSYCOPG2

Psycopg és l'adaptador de *PostgreSQL* més popular per al llenguatge de programació *Python*. En el seu nucli s'aplica plenament el *Python DB API* 2.0. Diverses extensions permeten l'accés a moltes de les característiques que ofereix *PostgreSQL*

3.4.5.- POSTGRESQL

PostgreSQL és un sistema de gestió de base de dades relacional i lliure, publicat sota la llicència *BSD*.

Com molts altres projectes de codi obert, el desenvolupament de *PostgreSQL* no és manejat per una empresa o una persona, sinó que és dirigit per una comunitat de desenvolupadors que treballen de forma desinteressada, altruista, lliure i recolzats per organitzacions comercials. Aquesta comunitat és denominada el *PGDG* (*PostgreSQL Global Development Group*).

Algunes de les seves principals característiques són:

Àmplia varietat de tipus natius

PostgreSQL proveeix nadivament suport per:

- Adreces MAC *(M*edia *A*ccess *C*ontrol*)*.
- Vectors.
- Text de llarg il·limitat.
- Nombres de precisió arbitrària.
- Blocs d'adreces estil *CIDR (C*lassless *I*nter-*D*omain *R*outing*)*.
- Figures geomètriques (amb una varietat de funcions associades)
- Adreces *IP (I*nternet *P*rotocol*)* (*IPv4* i *IPv6*).

Funcions

Blocs de codi que s'executen en el servidor. Poden ser escrits en diversos llenguatges, amb la potència que cadascun d'ells dóna, des de les operacions bàsiques de programació, tals com a bifurcacions i bucles, fins a les complexitats de la programació orientada a objectes o la programació funcional.

Les llançadores (*triggers* en anglès) són funcions enllaçades a operacions sobre les dades.

Alguns dels llenguatges que es poden usar són els següents:

- Un llenguatge propi anomenat *PL/PgSQL* (similar al *PL/SQL* d'oracle).
- *C*.
- *C++*.
- *Java PL/Java web*.
- *PL/Perl*.
- *PL/Python*.
- *PL/Ruby*.
- *PL/sh*.
- *PL/Tcl*.
- *PL/Scheme*.
- Llenguatge per a aplicacions estadístiques R per mitjà de *PL/R*.

PostgreSQL suporta funcions que retornen "files", on la sortida pot tractar-se com un conjunt de valors que poden ser tractats igual a una fila retornada per una consulta.

Les funcions poden ser definides per executar-se amb els drets del propi usuari o amb els drets d'un usuari prèviament definit. El concepte de funcions, en altres *DBMS*, són moltes vegades referides com a "procediments emmagatzemats" (*stored procedures* en anglès).

Triggers

Un llançador o *trigger* es defineix en una acció específica basada en alguna cosa ocurrent dins de la base de dades. En *PostgreSQL* això significa l'execució d'un procediment emmagatzemat basat en una determinada acció sobre una taula específica. Ara tots els llançadors es defineixen per sis característiques:

- El nom del llançador o *trigger*.
- El moment en què el llançador ha d'arrencar.
- L'esdeveniment del llançador haurà d'activar-se sobre...
- La taula on el llançador s'activarà.
- La freqüència de l'execució.
- La funció que podria ser cridada.

Llavors combinant aquestes sis característiques, *PostgreSQL* permet crear una àmplia funcionalitat a través del seu sistema d'activació de llançadors.

Alta concurrència	Mitjançant un sistema denominat *MVCC* (*Accés concurrent multiversió*) *PostgreSQL* permet a un procés escriure en una taula, mentre uns altres accedeixen a la mateixa taula sense necessitat de bloquejos. Cada usuari obté una visió consistent de l'últim al que se li va fer una assignació (*commit*). Aquesta estratègia és superior a l'ús de bloquejos per taula o per files comuna en altres bases, eliminant la necessitat de l'ús de bloquejos explícits.
Altres	Claus alienes també denominades Claus Foranes (*foreign keys*). Vistes. Integritat transaccional. Herència de taules. Tipus de dades i operacions geomètriques. Suport per a transaccions distribuïdes que permet a *PostgreSQL* integrar-se en un sistema distribuït format per diversos recursos (per exemple: una base de dades *PostgreSQL*, una altra *Oracle*, una cua de missatges *IBM MQ JMS* i un *ERP SAP*) gestionat per un servidor d'aplicacions on l'èxit ("commit") de la transacció global és el resultat de l'èxit de les transaccions locals.

Taula 4: Característiques de *PostgreSQL*

3.4.6.- DJANGO

En l'apartat 2.3 s'ha descrit breument què és *Django*. En aquest apartat s'explicarà com funciona i quines parts l'integren per poder entendre tots els passos que es descriuran en la part de configuracions.

En *Django* tenim els conceptes de projectes i aplicacions. Un projecte engloba diverses aplicacions i les fa treballar juntes per produir el resultat que veu l'usuari, generalment un portal web.

Com s'ha comentat abans, en *Django* hi ha projectes i aplicacions. Normalment treballem sobre les aplicacions (cadascuna en subdirectori dins del projecte) i quan l'aplicació està implementada l'activem dins del nostre portal associant-la a un *URL* editant el fitxer *urls.py* del projecte.

Les aplicacions normalment es componen dels següents elements:

- ***urls.py:*** En aquest fitxer s'especifica quines rutes de la *URL* van a cada funció *Python* que les implementa. *Django* ofereix un sistema de maneig d'*URL* basat en expressions regulars que associa una expressió regular a una vista. Per dissenyar les *URL* d'una aplicació *Django*, es construeix una mena de taula que mapeja patrons d'*URL* a funcions *Python* per executar (vistes). Amb això s'aconsegueix que les *URL* estiguin desacoblades de la resta de l'aplicació.

- ***views.py:*** En aquest fitxer és on estarà el codi que implementarà la "intel·ligència", que curiosament en *Django* en diuen "vistes" (en la majoria d'altres entorns *MVC* ho denominarien "controlador") o la part dinàmica de cada pàgina web. Hauria de validar que totes les dades del mateix són correctes. Generalment les vistes acaben retornant o bé un codi *HTTP* (com 404, 500, etc) o una plantilla .html a la qual se li passen una sèrie de dades.

- ***plantilles:*** les plantilles o *templates* són fitxers amb format *HTML* en els quals podem posar codi en el llenguatge de plantilles de *Django*. Encara que les plantilles poden ser perfectament fitxers *HTML* sense codi de plantilla, generalment les utilitzarem renderitzades al final d'una vista i amb certs valors que la vista li passa perquè els visualitzi.

- ***models.py***: aquí és on anem a definir els models que farem servir per les nostres dades, és a dir, les classes que representaran les nostres dades. En *Django* encara que podem accedir directament a la base de dades usant el llenguatge *SQL* gairebé mai ho farem així, sinó que definirem les nostres classes al *models.py* de cada aplicació i executarem "*manage.py syncdb*" perquè es "converteixi" en taules i registres *SQL* a la base de dades real. Després des del nostre codi (és a dir, des de les nostres vistes) farem servir aquestes dades com a objectes. Els models en *Django* es defineixen generalment creant variables de classe com registres (el tipus s'especifica amb una assignació al tipus correcte, per exemple `nom = models.CharField (max_length = 255)`).

Imatge 5: Funcionament Django

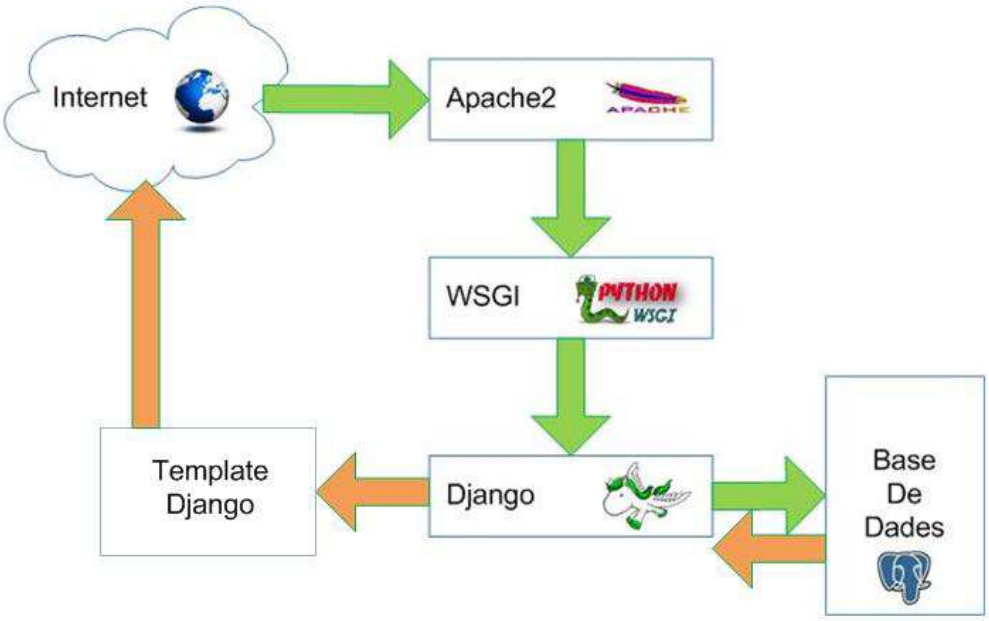
3.5.- INSTAL·LACIÓ, ESTRUCTURA I CONFIGURACIÓ

Per poder desenvolupar la nostra aplicació i testejar-la en un entorn de treball real el primer que hem d'entendre és que necessitem un servidor web capaç de comunicar-se amb *Django*. Per aquest fi tenim *WSGI*.

Tal com s'ha descrit anteriorment, *WSGI* és un estàndard de comunicació que utilitzen els servidors web per poder interpretar el codi dels *Frameworks* de *Python*. *WSGI* té una "limitació" i és que no pot servir el contingut estàtic (*js, imatges, css,* ..) només s'encarrega del contingut dinàmic.

Arribats a aquest punt veiem que hi ha una necessitat, la de poder dividir una petició en dues parts, una per a que la interpreti el servidor web (part estàtica), i una altra per passar-s'ho a *WSGI* (el contingut dinàmic). *Apache* és capaç de fer això sense cap problema.

A continuació es mostrarà un diagrama d'aquest funcionament:



Imatge 6: Diagrama de flux d'una petició des de Internet

Després de fer-nos una idea de com funcionen les peticions, anem a descriure els passos a seguir per una correcta instal·lació del programari necessari per desenvolupar el projecte. Aquesta tasca es pot dur a terme des de terminal (com s'ha fet en el nostre cas) o des de l'aplicació que facilita *Linux* per a la instal·lació de software, el *Synaptic*.

3.5.1.- INSTAL·LACIÓ

Si s'opta per fer-ho des de el *Synaptic*, els paquets necessaris que s'han de marcar per ser instal·lats són:

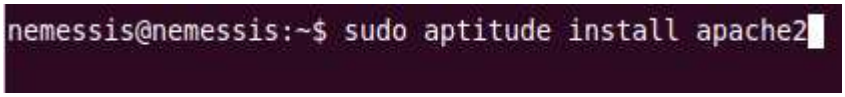
- *Apache2*
- *Libapache2-mod-wsgi*
- *python-django*
- *postgresql*
- *python-psycopg2*

Aquesta instal·lació però no es la recomanada ja que en molts casos podem no trobar les versions que nosaltres desitgem instal·lar. En el *Synaptic* trobem les últimes versions de programari que podrien no ser compatibles amb algun programari que tinguéssim instal·lat o no coincidir amb el que es desitja tenir instal·lat.

Si pel contrari s'opta per fer-ho des del terminal, a continuació es descriurà breument com fer-ho:

Primerament realitzarem la instal·lació d'*Apache2*. Per fer-ho escriurem en el terminal la comanda:

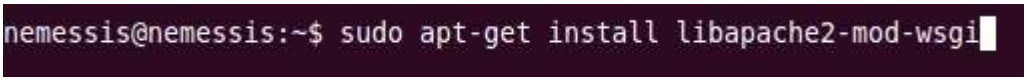
```
$ sudo aptitude install apache2
```



Imatge 7: Instal·lació d'Apache2

Després d'instal·lar el nostre servidor *Apache2*, haurem d'instal·lar *WSGI* per a la gestió de *Python* i del contingut dinàmic, per fer-ho escriurem en el terminal:

```
$ sudo apt-get install libapache2-mod-wsgi
```



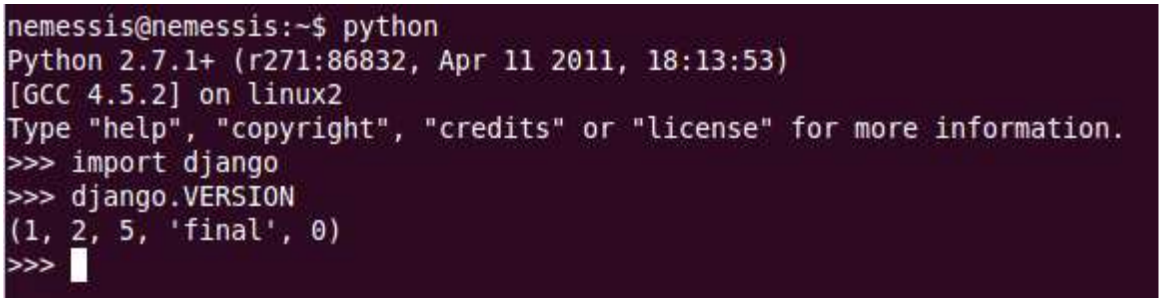
Imatge 8: Instal·lació del módul WSGI

Un cop fet això ja podem instal·lar *Django*.

Per fer-ho anirem a la pàgina oficial de *Django* (http://django.es/).

Seguirem el passos descrits a continuació:

```
Descarregar el paquet: Django-1.2.5.tar.gz
Descomprimir-lo: tar xzvf Django-1.2.5.tar.gz
Entrem a la carpeta: cd Django-1.2.5
Instal·lem Django: sudo python setup.py install
Després de seguir aquests passos, anem a verificar si la instal·lació
s'ha dut a terme correctament. Escriurem en el nostre terminal python
per executar l'intèrpret de Python, importarem django i consultarem la
versió instal·lada.
$ python
$ import django
$ django.VERSION
```



Imatge 9: Revisió instal·lació

En la imatge superior podem observar que a l'executar l'intèrpret de *Python*, ens diu la versió que tenim instal·lada al nostre sistema, si la versió no fos la desitjada hauríem d'instal·lar el *Python* corresponent.

Un cop tenim el *Django*, ja podem instal·lar *Postgresql* i *Psycopg2*, per fer-ho escriurem en el terminal:

```
$ sudo apt-get install postgresql9.0 python-psycopg2
```

Imatge 10: Instal·lació postgresql i psycopg2

Després d'instal·lar tot el programari necessari per poder operar, l'haurem de configurar per a que tingui el comportament desitjat.

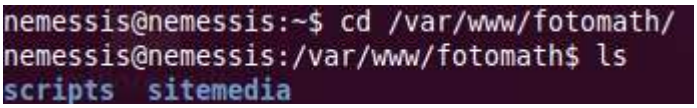
3.5.2.- ESTRUCTURA

En aquest apartat es descriurà l'estructura de directoris del projecte. Si és el primer cop que s'utilitza *Django* en l'equip, ens haurem de fer càrrec de la configuració inicial del sistema. El què és vol dir amb això, és que es necessita autogenerar el codi que defineix el projecte *Django*. Aquest codi és un conjunt de configuracions per una instància de *Django*, la qual inclou la configuració de la base de dades, opcions específiques de *Django* i detalls propis de la nostra aplicació.

Des de la línia d'ordres, s'utilitzarà *cd* per canviar de directori a aquell on es vulgui emmagatzemar el codi.

El primer a decidir és on volem allotjar el projecte. Es desaconsella posar els arxius de *Python* dins de l'arrel del nostre servidor web. S'ha decidit crear el directori *fotomath* dins de */var/www/fotomath* per utilitzar-lo com carpeta d'emmagatzematge.

A part dels fitxers normals del projecte, crearem 2 directoris: un contindrà els fitxers pertinents als *scripts* (aplicacions), i un altre anomenat *sitemedia* amb tot el contingut d'*imatges, css, java ....*



Imatge 11: Estructura fotomath

Dins el directori */var/www/* es crea *fotomath* amb la comanda:

```
$ mkdir fotomath
```

I dins el directori creat (*/var/www/fotomath/*) executarem la comanda:

```
$ django-admin.py startproject scripts
```

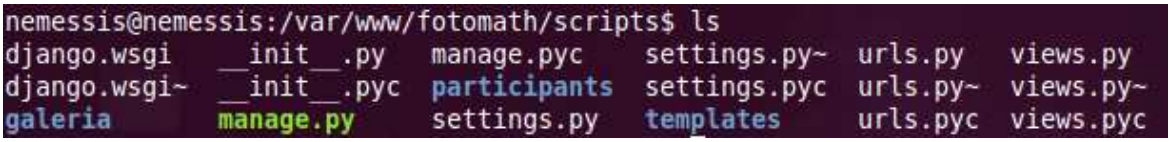
Aquesta comanda ens crearà:

scripts/

- *__init__.py*
- *manage.py*
- *settings.py*
- *urls.py*

Aquests arxius són:

- ***__init__.py:*** Un arxiu buit que li diu a *Python* que aquest directori hauria de ser considerat un paquet *Python*.
- ***manage.py:*** Una utilitat de línia de comandes que et permet interactuar de diferents formes amb aquest projecte *Django*.
- ***settings.py:*** Configuració per aquest projecte *Django*.
- ***urls.py:*** Les *URL* per aquest projecte *Django*; una "taula de continguts" del teu lloc basat en *Django*.



Imatge 12: Directori scripts

El directori de *sitemedia* es crearà dins el directori de *fotomath/*, aquest directori el crearem nosaltres amb la comanda:

```
$ mkdir sitemedia
```

I crearem també un directori per cada part, un per emmagatzemar el fitxer que contindrà els *css*, un altre pels arxius (*files*), un altre amb les imatges (*images*) i un altre amb el contingut *Javascript* (*js*).

sitemedia/

- *css/*
- *files/*
- *images/*
- *js/*

Imatge 13: Directori sitemedia

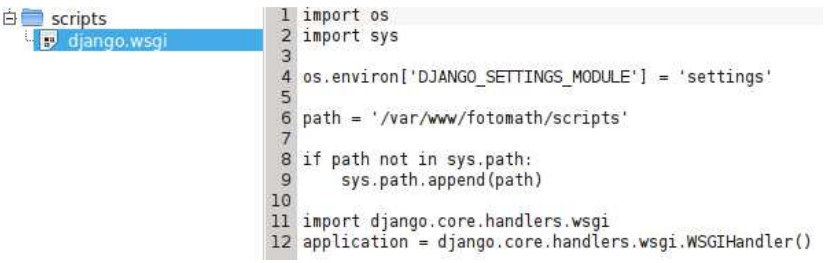
Després de creà l'estructura del projecte, anem a configurar-lo.

3.5.3.- CONFIGURACIÓ

En l'apartat anterior s'han descrit els passos a seguir per una correcta instal·lació del programari i s'ha començat a donar forma al nostre lloc. En aquest apartat es descriurà com configurar tot el programari instal·lat anteriorment.

3.5.3.1.- CONFIGURACIÓ DE WSGI

Per configurar *WSGI*, el que s'ha de fer és afegir un fitxer de configuració dins el directori *scripts/* aquest fitxer l'anomenarem *django.wsgi* amb el contingut següent:



Imatge 14: Django.wsgi

D'aquesta manera *WSGI* sap on ha d'anar a buscar el fitxer de configuració *Settings* de la nostra aplicació.

3.5.3.2.- CONFIGURACIÓ DE POSTGRESQL I DJANGO

Després d'afegir el fitxer de configuració de *WSGI*, anem ara a configurar la base de dades i relacionar-la amb *Django*.

És recomana que abans de ficar-se a configurar *PostgreSQL*, es creï el projecte de *Django* al que ha d'anar relacionat.

Dins de l'estructura d'arxius creats amb la comanda s'editen els mòduls, *urls* i *Settings*:

```
$ django-admin startproject scripts
```

Urls representa una taula de contingut de totes les pàgines que generem amb *Python*:

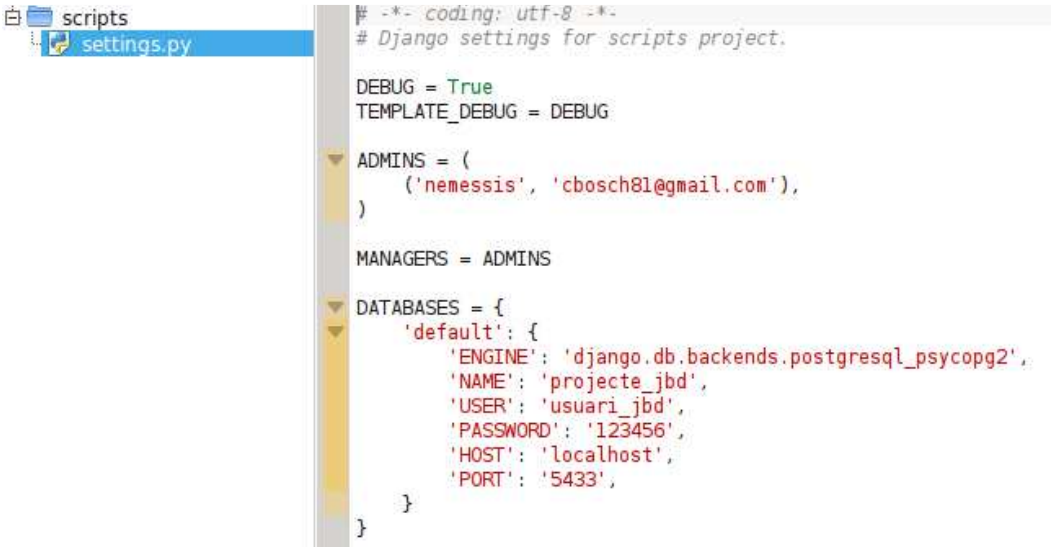
- *http://fotomath/formulari/*

És molt recomanable activar la funció d'administració del lloc. Per activar aquesta funció d'administració el que cal fer es modificar l'arxiu *urls.py* i descomentar, treure el coixinet, assignant valors a les variables.

Una mostra de com podria quedar l'arxiu en la imatge següent:

Imatge 15: Urls

Ara anem al nostre *settings* on configurem algunes variables. Per una millor comprensió l'hem dividit en diferents parts però totes formen part del mateix fitxer.



Imatge 16: Settings 1

En el *Settings 1*, definim la variable ADMIN amb el nom de l'administrador del projecte o administradors i la seva adreça de correu, pot ser convenient per problemes que puguin sorgir després o per notificacions al/s administrador/s.

En les següents línies definim les variables del gestor de base de dades, la nostra base de dades esta feta amb *PostgreSQL*.

Definim les variables 'ENGINE' per indicar el tipus de base de dades que utilitzarem, 'NAME' per definir el nom de la nostra base de dades, 'USER' per indicar l'usuari en la base de dades, 'PASSWORD' per definir la clau de la base de dades, 'HOST' el declarem com a localhost ja que serà local i 'PORT' per indicar el port de les comunicacions.

```
# Local time zone for this installation. Choices can be found here:
# http://en.wikipedia.org/wiki/List_of_tz_zones_by_name
# although not all choices may be available on all operating systems.
# If running in a Windows environment this must be set to the same as your
# system time zone.
TIME_ZONE = 'Europe/Andorra'

# Language code for this installation. All choices can be found here:
# http://www.i18nguy.com/unicode/language-identifiers.html
LANGUAGE_CODE = 'ca'
```

Imatge 17: Settings 2

En aquesta segona part definim el TIME_ZONE que ens indica on es troba ubicada l'aplicació. En el nostre cas és *'Europe/Andorra'*. LANGUAGE_CODE ens serveix per definir l'idioma, que és català (ca).

```
# URL prefix for admin media -- CSS, JavaScript and images. Make sure to use a
# trailing slash.
# Examples: "http://foo.com/media/", "/media/".
ADMIN_MEDIA_PREFIX = '/adminmedia/'

# Make this unique, and don't share it with anybody.
#SECRET_KEY = '6#_u#^_ks&-6qhgtww8p5zlyz8n6to6sqn#pjbxs(ko^vjb)8j'

# List of callables that know how to import templates from various sources.
TEMPLATE_LOADERS = (
    'django.template.loaders.filesystem.load_template_source',
    'django.template.loaders.app_directories.load_template_source',
#    'django.template.loaders.filesystem.Loader',
#    'django.template.loaders.app_directories.Loader',
#     'django.template.loaders.eggs.load_template_source',
)

MIDDLEWARE_CLASSES = (
    'django.middleware.common.CommonMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
)

ROOT_URLCONF = 'urls'

TEMPLATE_DIRS = (
    '/var/www/fotomath/scripts/templates',
    # Put strings here, like "/home/html/django_templates" or "C:/www/django/templates".
    # Always use forward slashes, even on Windows.
    # Don't forget to use absolute paths, not relative paths.
)

CAPTCHA_CHALLENGE_FUNCT = 'captcha.helpers.math_challenge'
CAPTCHA_FONT_SIZE = 35
CAPTCHA_LETTER_ROTATION = 0
CAPTCHA_FOREGROUND_COLOR = '#006600'


INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.admin',
    'captcha',
    'participants',
)
```

Imatge 18: Settings 3

En la tercera part del *Settings* definirem les aplicacions que es creen per al nostre projecte. Si creem una aplicació anomenada *participants* l'hem d'incloure dins de INSTALLED_APPS. Escrivim l'aplicació d'administrador en aquesta llista també.

Després de deixar configurat el fitxer amb les dades desitjades, només queda quadrar-ho amb la base de dades. Per fer-ho, modificarem l'arxiu de configuració *pg_hba.conf*.

```
nemessis@nemessis:/etc/postgresql/9.0/main$ ls
environment  pg_hba.conf   pg_ident.conf    start.conf
pg_ctl.conf  pg_hba.conf~  postgresql.conf
```

Imatge 19: Main postgresql

En el terminal entrem la següent comanda:

```
$ etc/postgresql/9.0/main/pg_hba.conf
```

Al final de l'arxiu trobem l'opció per dir-li a *PostgreSQL* si ha de demanar o no *password*. On diu *md5* hi col·loquem *trust*, menys en l'última línia. Guardem l'arxiu i reiniciem *PostgreSQL*.

```
# Database administrative login by UNIX sockets
local   all         postgres                          trust

# TYPE  DATABASE        USER            CIDR-ADDRESS            METHOD

# "local" is for Unix domain socket connections only
local   all             all                                     trust
# IPv4 local connections:
host    all             all             127.0.0.1/32            trust
# IPv6 local connections:
host    all             all             ::1/128                 md5
```

Imatge 20: Pg_hba.conf

La instal·lació del *PostgreSQL* crea per defecte un usuari '*postgres*' que és el superusuari pel servidor de la base de dades. La primera cosa que cal fer és canviar la contrasenya (*password*) d'aquest usuari. Això requereix canviar-ho tant dins del postgres com al sistema operatiu. Tot i que no és necessari resulta convenient que les dues contrasenyes coincideixin. És convenient fer-ho en l'ordre que descriurem ja que si no ho fem així no ens deixarà canviar el *password* en el segon.

Per canviar la contrasenya a l'usuari *postgres* del sistema operatiu, en el terminal escriurem:

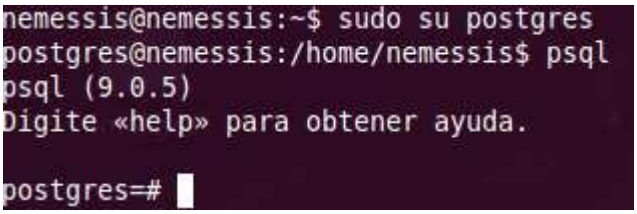
```
$ sudo passwd -d postgres
$ sudo su postgres -c passwd
```

```
nemessis@nemessis:~$ sudo passwd -d postgres
[sudo] password for nemessis:
passwd: se ha cambiado la información de expiración de la contraseña
nemessis@nemessis:~$ sudo su postgres -c passwd
Introduzca la nueva contraseña de UNIX:
Vuelva a escribir la nueva contraseña de UNIX:
passwd: contraseña actualizada correctamente
```

Imatge 21: Password postgres S.O

Després en el terminal també, escriurem:

```
$ sudo su postgres
$ psql
postgres = #
```

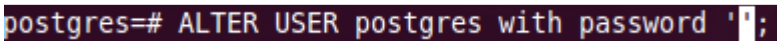


Imatge 22: Configuració postgresql

Aquesta comanda ens ha fet entrar dins de *PostgreSQL* (*psql*), on queda el *#*, escrivim

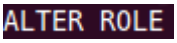
```
postgres = # ALTER USER postgres with password ‘nou’;
```

On *nou* és el nou *password* de l’usuari *postgres*.



Imatge 23: Usuari postgresql

Si aquesta ordre s’ha dut a terme correctament, apareixerà a sota *ALTER ROLE*.



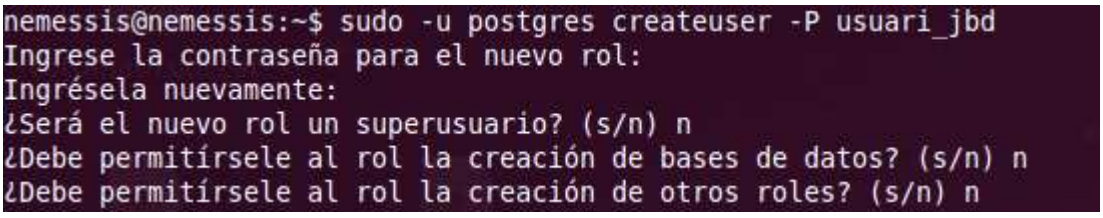
Imatge 24: Alter role

Sortim de *PostgreSQL* amb *\q.*

Ara creem un nou usuari per a la nostra aplicació *Django*, que s’utilitzarà per entrar a la base de dades. A aquest usuari no li donem cap dels privilegis que té el superusuari per crear bases de dades o rols.

En el terminal, escrivim:

```
$ sudo -u postgres createuser -P usuari_jbd
```



Imatge 25: Crea usuari per la base de dades

L’ *usuari_dbd* és l’usuari de la base de dades que utilitzarem per connectar-nos des de l’aplicació *Django*. Cal recordar que el nom de l’usuari i la contrasenya són els que hem utilitzat en el fitxer *settings.py* de la nostra aplicació *Django*.

Després, creem una base de dades anomenada *projecte_jbd*. Ens cal crear una base de dades per utilitzar-la amb la nostra aplicació *Django*.

Entrem dins la interfície de *psql* i escrivim:

```
# CREATE DATABASE projecte_jdb OWNER usuari_jbd ENCODING ‘UTF8’;
```

```
postgres=# CREATE DATABASE projecte_jbd owner usuari_jbd encoding 'UTF8';
CREATE DATABASE
```

Imatge 26: Create database

Això crea una base de dades anomenada *projecte_jbd* propietat del *usuari_jbd*.

Si la base de dades ja està creada ha d’aparèixer un error com el següent:

```
postgres=# CREATE DATABASE projecte_jbd;
ERROR:  la base de datos «projecte_jbd» ya existe
```

Imatge 27: Error create database

Reiniciem PostgreSQL.

Bé, és hora de provar que *PostgreSQL* funciona amb *Django*. Entrem per terminal allí on es troba el nostre projecte de *Django*. Quan hi som, li donem la comanda següent per sincronitzar el nostre projecte amb *PostgreSQL*:

```
$ python manage.py syncdb
```

```
nemessis@nemessis:/var/www/fotomath/scripts$ python manage.py syncdb
```

Imatge 28: Sincronitzar la base de dades

L'execució d'aquesta comanda ens sincronitza la base de dades que hem creat a l'arxiu models.py. Ha de mostrar el següent:

```
nemessis@nemessis:/var/www/fotomath/scripts$ sudo python manage.py syncdb
Creating table auth_permission
Creating table auth_group_permissions
Creating table auth_group
Creating table auth_user_user_permissions
Creating table auth_user_groups
Creating table auth_user
Creating table auth_message
Creating table django_content_type
Creating table django_session
Creating table django_site
Creating table django_admin_log
Creating table captcha_captchastore
Creating table participants_concursant
Creating table participants_imatge
Creating table participants_jurat
Creating table participants_puntsimatge

You just installed Django's auth system, which means you don't have any superuse
rs defined.
Would you like to create one now? (yes/no):
```

Imatge 29: Sincronització de la base de dades 1

En aquesta segona part se'ns pregunta si volem crear un superusuari, aquest ens serveix per administrar la web de *Django*. En cas afirmatiu li assignen el nom de l'usuari, el correu i la contrasenya.

```
Would you like to create one now? (yes/no): yes
Username (Leave blank to use 'root'): nemessis
E-mail address: cbosch81@gmail.com
Password:
Password (again):
Superuser created successfully.
Installing index for auth.Permission model
Installing index for auth.Group_permissions model
Installing index for auth.User_user_permissions model
Installing index for auth.User_groups model
Installing index for auth.Message model
Installing index for admin.LogEntry model
Installing index for participants.Imatge model
Installing index for participants.Puntsimatge model
No fixtures found.
```

Imatge 30: Sincronització de la base de dades 2

Si hem realitzat correctament els passos anteriors, en executar-ho en el nostre navegador:

- http://fotomath/admin

Ha d'aparèixer la següent imatge:

Imatge 31: Imatge de l'apartat d'Administració

On hi fica *usuari* hem d'escriure l'usuari que hem creat al executar la comanda de:

```
$ python manage.py syncdb
```

La contrasenya també serà la mateixa.

Escriure correctament aquets dos passos ens donarà accés a la part administrativa de la nostra aplicació.

Aquesta eina és molt poderosa ja que podrem administrar les nostres aplicacions d'una manera fàcil i àgil.

3.5.3.3.- CONFIGURACIÓ DEL SISTEMA

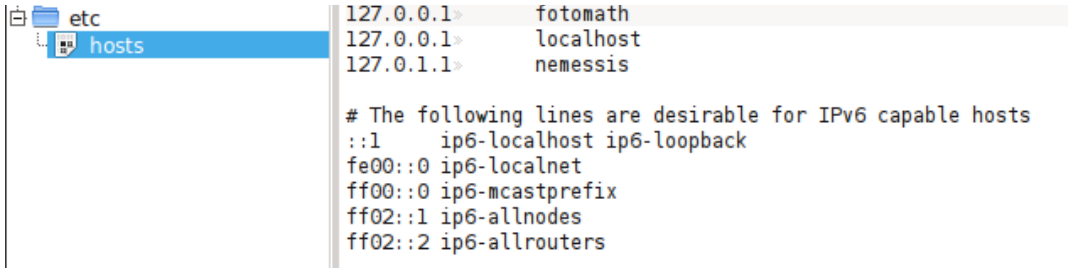
Per configurar el sistema i que ens mostri la nostra aplicació, hem de modificar l'arxiu:

```
$ /etc/hosts
```

I afegir:

```
127.0.0.1       fotomath
```

D'aquesta manera el sistema sap on s'ha d'executar *fotomath*, que en el nostre cas és la màquina de proves.



Imatge 32: Hosts

3.5.3.4.- CONFIGURACIÓ APACHE2 EN UBUNTU

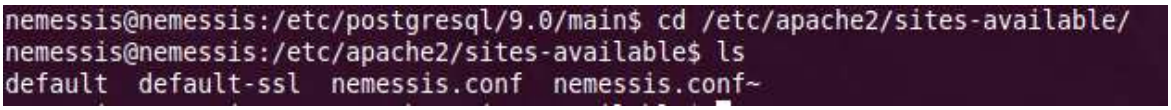
Després de dur a terme la configuració de la base de dades i de *Django*, per adequar-ho a les nostres necessitats, anem a configurar el servidor Apache2.

Un dels primers passos a seguir es enllaçar *WSGI* amb el nostre servidor Apache2. Per fer-ho, hem de crear el següent enllaç simbòlic: (Sempre en una distribució Ubuntu o Debian)

```
$ ln -s /etc/apache2/mods-available/wsgi.load/etc/apache2/mods-
enabled/wsgi.load
```

Després en el directori:

```
$ /etc/apache2/sites-available/
```



Imatge 33: Sites available

Hem d'afegir el fitxer *nemessis.conf* amb el contingut que es mostra en la imatge que segueix:

```
<VirtualHost *:80>
    ServerName nemessis

    DocumentRoot /var/www/fotomath

    Alias /sitemedia /var/www/fotomath/sitemedia
    Alias /2009 /var/www/fotomath/2009

    <Directory /var/www/fotomath/sitemedia>
        Order deny,allow
        Allow from all
    </Directory>

    <Directory /var/www/fotomath/2009>
        Order deny,allow
        Allow from all
    </Directory>

    Alias /adminmedia /usr/share/pyshared/django/contrib/admin/media

    <Directory /usr/share/pyshared/django/contrib/admin/media>
        Order allow,deny
        Allow from all
    </Directory>

    <IfModule mod_wsgi.c>
        WSGIScriptAlias / /var/www/fotomath/scripts/django.wsgi
    </IfModule>

    <Directory /var/www/fotomath/scripts>
        Order allow,deny
        Allow from all
    </Directory>

    ErrorLog /var/log/apache2/fotomath-error.log
    LogLevel warn
    CustomLog /var/log/apache2/fotomath-access.log combined
</VirtualHost>
```

Imatge 34: Arxiu configuració Apache2

A continuació eliminarem els enllaços simbòlics que hi hagi al directori:

```
$/etc/apache2/sites-enabled/
```

I farem l'enllaç simbòlic següent:

```
$ ln –s /etc/apache2/sitesavailable/elmeudomini.conf/etc/apache2/sites-enabled/000-elmeudomini.conf
```

```
nemessis@nemessis:/etc/apache2/sites-enabled$ ls
000-nemessis.conf
```

Imatge 35: Enllaç simbòlic

El fitxer

```
$ /etc/apache2/httpd.conf
```

Ha de tenir només la línia següent:

```
ServerName localhost
```

Imatge 36: Httpd.conf

I el fitxer

```
$ /etc/apache2/ports.conf
```

Ha de contenir:

```
# If you just change the port or add more ports here, you will likely also
# have to change the VirtualHost statement in
# /etc/apache2/sites-enabled/000-default
# This is also true if you have upgraded from before 2.2.9-3 (i.e. from
# Debian etch). See /usr/share/doc/apache2.2-common/NEWS.Debian.gz and
# README.Debian.gz

NameVirtualHost *:80
Listen 80

<IfModule mod_ssl.c>
    # If you add NameVirtualHost *:443 here, you will also have to change
    # the VirtualHost statement in /etc/apache2/sites-available/default-ssl
    # to <VirtualHost *:443>
    # Server Name Indication for SSL named virtual hosts is currently not
    # supported by MSIE on Windows XP.
    Listen 443
</IfModule>

<IfModule mod_gnutls.c>
    Listen 443
</IfModule>
```

Imatge 37: Ports.conf

Finalment, hem de comprova que *l'Apache* està ben configurat. Per això el reiniciarem:

```
$ sudo /etc/init.d/apache2 restart
```

```
nemessis@nemessis:~$ sudo /etc/init.d/apache2 restart
 * Restarting web server apache2
 ... waiting                                                    [ OK ]
```

Imatge 38: Reinici d'Apache2

En aquest capítol s'exposarà com s'ha desenvolupat i aplicat tot el codi necessari per l'aplicació del formulari web. A més, s'hi afegeixen imatges del resultat final.

Tant en el formulari web com en la gestió d'imatges primer és treballarà en un servidor local de proves i un cop acabades les proves és passaran els arxius i mòduls al servidor definitiu. Aquest pas queda detallat en el punt 6. Servidor FotoMath

4.- APLICACIÓ 1: FORMULARI WEB

El primer que es va haver de fer per poder desenvolupar l'aplicació del formulari, va ser parlar amb els encarregats del concurs per veure quins camps necessitaven que contingues el formulari. Després de la reunió amb els membres, es va creure convenient que el formulari contés amb quatre camps pel participant (nom, primer cognom, segon cognom i adreça de correu) i un màxim de 5 camps per l'enviament de fotografies.

Aquesta part es bàsica en qualsevol desenvolupament web, s'han de conèixer les necessitats dels participants i dels membres del concurs per poder desenvolupar una aplicació que sigui la més adient.

Un cop decidits quins camps ha de contenir el formulari, ja es pot començar a desenvolupar l'aplicació. Ja hem vist en la part d'instal·lació i configuració l'estructura que se li ha donat al projecte. Així, en la carpeta dels scripts es crea una nova aplicació anomenada *participants* que contindrà el formulari i els arxius relacionats.

Per crear l'aplicació dels *participants* o qualsevol aplicació en *Django*, s'ha d'executar la comanda:

```
$ python manage.py startapp participants
```

```
nemessis@nemessis:/var/www/fotomath/scripts$ python manage.py startapp participants
```

Imatge 39: Creació d'una nova aplicació

Aquesta comanda ens crea dins el projecte una aplicació anomenada *participants* amb els següents arxius:

participants/

- *__init__.py*
- *models.py*
- *tests.py*
- *views.py*

```
nemessis@nemessis:/var/www/fotomath/scripts$ cd participants
nemessis@nemessis:/var/www/fotomath/scripts/participants$ ls
admin.py   extra.py   forms.py    __init__.py   models.py~  proves~     views.py
admin.py~  extra.py~  forms.py~   __init__.pyc  models.pyc  tests.py    views.py~
admin.pyc  extra.pyc  forms.pyc   models.py     proves      tests.pyc   views.pyc
```

Imatge 40: Contingut de l'aplicació participants

Quina és la diferència entre un projecte i una aplicació?

Una aplicació és un programa web que fa alguna cosa - per exemple, una bitàcola, un registre de dades públiques o un sistema d'enquestes.

Un projecte és un conjunt d'aplicacions configurades per un lloc web particular. Un projecte pot contenir múltiples aplicacions. Una aplicació pot pertànyer a múltiples projectes.

Aquesta aplicació creada l'haurem d'afegir al nostre *settings.py* (fitxer de configuracions) dins de l'apartat *INSTALLED_APPS*, Aquesta variable conté el nom de totes les aplicacions *Django* que estan activades en aquesta instància de *Django*.

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.admin',
    'captcha',
    'participants',
)
```

Imatge 41: Settings

4.1.- MODELS.PY

El primer pas per codificar una aplicació web amb Django és definir els teus models, l'estructura de la base de dades, amb metadades addicionals.

Un model és la font única i definitiva d'informació de les teves dades. Conté els camps i especificacions de les dades que s'estan emmagatzemant.

Django segueix el Principi DRY (Your say Repeat Yourself - No repeteixis). L'objectiu és definir el model de dades en un sol lloc i deduir coses automàticament a partir d'ell.

En el nostre sistema de formulari crearem 4 models:

- Concursant,
- Imatge,
- Jurat i
- Punts Imatge.

Aquests models es representen amb 4 classes *Python*.

El codi és directe. Cada model està representat per una subclasse de *django.db.models.Model*. Cada model té algunes variables de classe, que representen camps a la base de dades.

Cada camp està representat per una instància d'una classe *models.Field*, per exemple, *models.CharField* per caràcters i *models.DateTimeField* per datetimes. D'aquesta manera se li diu a *Django* quin tipus de dades conté cada camp.

El nom de cada instància de *models.Field* (per exemple nom o cognom1 o cognom2...) és el nom del camp, en un format agradable per a la base de dades. Aquest valor serà usat en el nostre codi *Python* i la base de dades el farà servir com el nom de la columna corresponent.

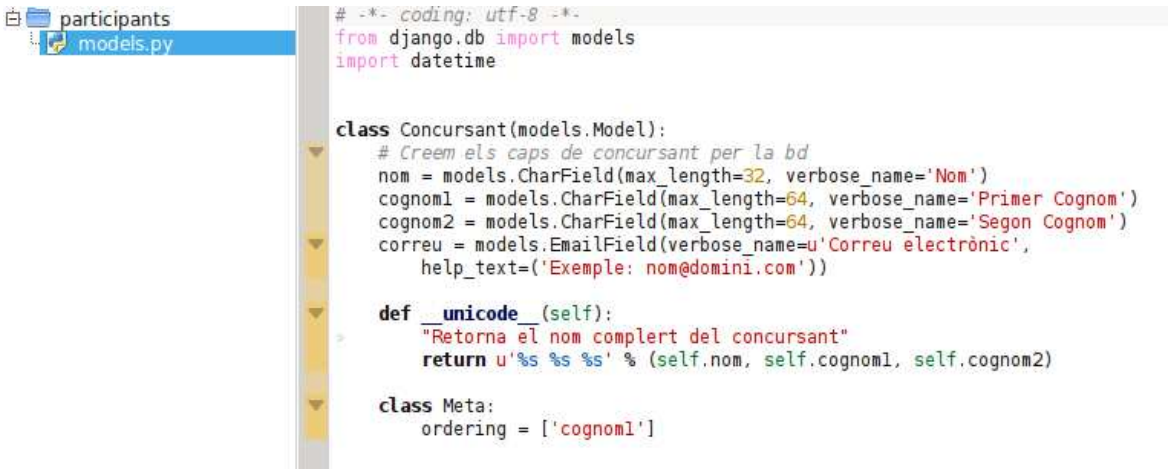
Podem fer servir com a primer argument d'un *Field* un nom més llegible i serveix com a documentació. Si no es lliura aquest camp, *Django* farà servir el nom del camp a la classe.

Algunes classe *Field* tenen elements obligatoris. Per exemple, *CharField* requereix que se li lliuri un atribut *max_length*, no només en relació a la base de dades, sinó també a l'hora de fer validacions, com veurem després.

Finalment, queda definir una relació usant *models.ForeignKey*. Això informa a *Django* que cada classe esta relacionada amb l'anterior. *Django* permet totes les relacions de base de dades típiques: molts a un, molts a molts i un a un.

Després d'aquesta breu explicació anem a entrar dins de cada classe més a fons. Per fer-ho editarem l'arxiu creat en blanc *models.py* i escriurem el nostre codi *Python*.

4.1.1.- CLASSE CONCURSANT



Imatge 42: Models 1

La nostra classe principal és la classe *Concursant*, les altres estaran relacionades amb ella.

En aquesta classe hem definit els caps de la nostra base de dades tal com es mostra en la imatge anterior.

- ***Nom***: com un *CharField* amb una longitud màxima de 32 caràcters i el nom que volem que aparegui al formulari. Està definit pel *verbose_name* com a "*Nom*".
- ***Cognom1 i cognom2***: també estan definits com a *CharField*, amb una longitud màxima de 64 caràcters i amb "*Primer Cognom*" i "*Segon Cognom*" com a mostra per a que aparegui al formulari.
- ***Correu***: esta definit com un *EmailField*, ja que el que esperem rebre és una adreça de correu electrònic amb un format vàlid i no una cadena de caràcters com en els anteriors casos. L'hem definit com a "*Correu electrònic*" per a que aparegui al formulari. El *help_text*, l'utilitzem per a que aparegui, al costat del requadre on s'ha d'escriure, una ajuda per a que l'usuari sàpigui quin format de dades s'espera rebre.

Hem de tenir clar que *nom*, *cognom1*, *cognom2* i *correu* són els camps de la base de dades amb la longitud i característiques que li hem assignat. Si no haguéssim emprat el *verbose_name*, *Django* hauria utilitzat el nom del camp de la base de dades per mostrar-lo al formulari.

La següent instrucció que podem observar és el *def __unicode_(self)*. Amb aquesta instrucció li estem dient a *Django* que ens ha de retornar tres cadenes de caràcters (strings %s %s %s): el nom, cognom1 i cognom2.

Per finalitzar la classe *Concursant*, veiem la classe Meta on li diem a *Django* com ha d'ordenar el que li hem dit que ens retorni en la instrucció anterior. En aquest cas ho ordenem per *cognom1*.

Aquest camp es bastant útil per la part administrativa ja que permet visualitzar els concursants ordenats pel primer cognom.

4.1.2.- CLASSE IMATGE

```
class Imatge(models.Model):
    # Creem els caps de imatge per la bd
    concursant = models.ForeignKey(Concursant)
    nom_imatge = models.CharField(max_length=128, verbose_name=u'Títol de la imatge')
    imatge = models.ImageField(upload_to='images/photos/2011', verbose_name='Adjunta la imatge')
    creada = models.DateTimeField(auto_now_add=True)
    modificada = models.DateTimeField(auto_now=True)

    def __unicode__(self):
        "Retorna el nom de la imatge"
        return u'%s' % (self.nom_imatge)

    class Meta:
        ordering = ['nom_imatge']
```

Imatge 43: Models 2

La segona classe que hem definit al nostre model, és la classe *Imatge*.

Aquesta classe depèn de la classe *Concursant* i n'és una clau forana. Per fer aquesta assignació s'utilitza el *ForeignKey(Concursant)*. Es tracta d'una relació un a un ja que una imatge només pot estar relacionada a un concursant encara que un concursant pot tenir més d'una imatge, essent totes elles imatges diferents.

Després de fer la relació entre classes definim els camps que ha de tenir la classe *Imatge*.

- ***Nom_imatge:*** la definim com a *CharField* ja que esperem una cadena de caràcters amb una longitud màxima de 128 i "*Títol de la imatge*" és el que volem que ens aparegui al formulari.
- ***Imatge:*** és un camp *ImageField*, ja que el que esperem rebre és una imatge, un arxiu que l'usuari adjuntarà i nosaltres hem de guardar. El lloc on l'hem de guardar el definim amb el *upload to=*. La carpeta on ho guardarem és la *images/photos/2011*. Aquesta carpeta la creem nosaltres dins el nostre projecte, dins *sitemedia/images*. El text d'ajuda que es fica és "*Adjunta la imatge*", per a que l'usuari vegi què és el que ha de fer.

Els següents dos camps de *creada* i *modificada* són dos camps de control que es va creure oportú d'afegir. Com el contingut serà una data, són camps *DataTimeField*. A l'hora de definir-los hi ha una diferencia entre els dos camps: *auto_now_add* fica la data de creació de l'arxiu però no permet modificar el camp; *auto_now* permet modificacions, així és pot tenir controlat si s'havia de fer algun canvi en la imatge, ja sigui per mida o format.

En la definició li estem dient a *Django* que ens ha de retornar el nom de la imatge, la següent classe meta en aquest cas no faria falta ja que li estem dient que ordeni per nom de la imatge i només estem retornant un camp. Tot i això es creu oportú ficar-ho per si es fa alguna modificació i volem que retorni algun camp més.

4.1.3.- CLASSE JURAT

```
class Jurat(models.Model):
    # Creem els caps de jurat per la bd
    imatge = models.ManyToManyField(Imatge, through='Puntsimatge')
    nom = models.CharField(max_length=128, verbose_name='Nom de la persona del jurat',
        help_text=(u'Escriu el teu nom com a jurat'))

    def __unicode__(self):
        "Retorna el nom de la persona del jurat"
        return u'%s' % (self.nom)

    class Meta:
        ordering = ['nom']
```

Imatge 44: Models 3

En aquesta tercera classe definida en el nostre model observem un canvi.

La classe *Jurat* té una relació amb la classe *Imatge* de molts a molts. Això és així ja que una imatge pot tenir diferents jurats i cada jurat tindrà moltes imatges, aquesta assignació la fem amb *ManyToManyField*.

La segona assignació que podem observar és el *nom* de la persona de jurat, amb una longitud màxima de 128 caràcters, amb un text per mostrar de "*Nom de la persona del jurat*" i una ajuda de "*Escriu el teu nom com a jurat*".

La classe *Jurat* retorna el nom de la persona del jurat i ho ordena per nom.

4.1.4.- CLASSE PUNTS IMATGE

```
class Puntsimatge(models.Model):
    # Creem els caps de puntsimatge per la bd
    imatge = models.ForeignKey(Imatge)
    jurat = models.ForeignKey(Jurat)
    punts = models.IntegerField(max_length=2, verbose_name='Puntuació de la imatge',
        help_text=(u'Enter 2 dígits sense decimals'))

    def __unicode__(self):
        "Retorna les puntuacions de les imatges"
        return u'%s' % (self.punts)

    class Meta:
        ordering = ['punts']
```

Imatge 45: Models 4

Per finalitzar el nostre *models*, tenim la classe *Puntsimatge*.

Aquesta classe depèn de *Imatge* i *Jurat*. Com es troba relacionada amb les dues classes anteriors hem de fer dues assignacions de clau forana, *ForeignKey*, una per la classe *Imatge* i una altre per *Jurat*. Aquesta assignació serà un a un ja que una nota només pot estar relacionada amb una imatge i aquesta només l'ha ficat una persona del jurat. Així, poden coincidir dues notes de diferents membres del jurat.

Després de fer aquesta assignació definim *punts* com un *IntegerField* ja que el que esperem rebre és un número, amb una longitud màxima de camp de 2 caràcters. Com a text per a mostrar al formulari s'ha elegit "*Puntuació de la imatge*" i com a text ajuda "*Enter 2 dígits sense decimals*". El què ens retorna aquesta classe són els *punts* de la imatge i ordenats per *punts*.

4.1.5.- SINCRONITZACIÓ AMB LA BASE DE DADES

Ara que ja tenim creada l'estructura de la nostra base de dades, crearem les taules a la base de dades sincronitzant-la amb *Django*. Per fer-ho executarem la comanda:

```
nemessis@nemessis:/var/www/fotomath/scripts$ sudo python manage.py syncdb
```

Imatge 46: syncdb

Aquesta comanda fa una sincronització del *models* amb la base de dades i les taules que no estan creades les crea. S'ha de tenir en compte que el *models* és una de les parts més importants i que fer una modificació al *models* significa modificar la base de dades i el seu format. Alguns canvis als *models* poden obligar a tornar a crear la base de dades per evitar errors.

Per observar les taules creades i la seva estructura podem executar la comanda:

```
nemessis@nemessis:/var/www/fotomath/scripts$ python manage.py sql participants
```

Imatge 47: sql participants

En executar aquesta comanda es mostra l'estructura de la base de dades, amb els seus noms, valors i característiques. Podrem observar l'estructura que hem creat amb el models per la nostra base de dades en la imatge següent:

```
nemessis@nemessis:/var/www/fotomath/scripts$ python manage.py sql participants
BEGIN;
CREATE TABLE "participants_concursant" (
    "id" serial NOT NULL PRIMARY KEY,
    "nom" varchar(32) NOT NULL,
    "cognom1" varchar(64) NOT NULL,
    "cognom2" varchar(64) NOT NULL,
    "correu" varchar(75) NOT NULL
)
;
CREATE TABLE "participants_imatge" (
    "id" serial NOT NULL PRIMARY KEY,
    "concursant_id" integer NOT NULL REFERENCES "participants_concursant" ("id") DEFERRABLE INITIALLY DEFERRED
,
    "nom_imatge" varchar(128) NOT NULL,
    "imatge" varchar(100) NOT NULL,
    "creada" timestamp with time zone NOT NULL,
    "modificada" timestamp with time zone NOT NULL
)
;
CREATE TABLE "participants_jurat" (
    "id" serial NOT NULL PRIMARY KEY,
    "nom" varchar(128) NOT NULL
)
;
CREATE TABLE "participants_puntsimatge" (
    "id" serial NOT NULL PRIMARY KEY,
    "imatge_id" integer NOT NULL REFERENCES "participants_imatge" ("id") DEFERRABLE INITIALLY DEFERRED,
    "jurat_id" integer NOT NULL REFERENCES "participants_jurat" ("id") DEFERRABLE INITIALLY DEFERRED,
    "punts" integer NOT NULL
)
;
COMMIT;
```

Imatge 48: sql participants 2

Després de definir la nostra base de dades, els camps que ha de tenir i el formulari, anem a definir la part administrativa. Aquesta part és molt útil per gestionar el lloc i per fer proves de funcionament.

4.2.- ADMIN.PY

Per fer aquesta part administrativa, hem de crear un nou fitxer al nostre projecte. En el nostre cas s'ha anomenat *admin.py*.

Per explicar millor allò que s'ha fet i perquè s'ha fet s'ha desglossat l'explicació en diferents parts.

4.2.1.- CLASSE PUNTSIMATGEINLINE

```
# -*- coding: utf-8 -*-
from django.contrib import admin
from participants.models import Concursant, Imatge, Jurat, Puntsimatge


class PuntsimatgeInline(admin.TabularInline):
    # Definim la class puntsimatge per la part administrativa
    model = Puntsimatge
    extra = 1

admin.site.register(Puntsimatge)
```

Imatge 49: admin1

En les primeres línies podem observar dos *from x import y*.

Tenim que de *django.contrib* (un fitxer de *Django*) importem *admin*. Això ho fem per poder utilitzar algunes variables de la part administrativa.

El segon *import*, el fem des del nostre *participants.models* i importem les classes *Concursant, Imatge, Jurat i Puntsimatge* per poder-les utilitzar al fitxer de la part administrativa.

La classe *PuntsimatgeInline* la definim per poder administrar els punts, la valoració numèrica de cada imatge. Així, el jurat podrà valorar les imatges des de la part administrativa. Li diem que és del tipus *TabularInline* per poder afegir més notes utilitzant el *extra=1*. Tot això ho guardem amb *admin.site.register(Puntsimatge)*.

4.2.2.- CLASSE IMATGEINLINE

```
class ImatgeInline(admin.TabularInline):
    # Definim la class imatge i li donem atributs per a poder adjuntar fins a 5 imatges
    model = Imatge
    max_num = 5
    extra = 1
```

Imatge 50: admin2

La classe *ImatgeInline* també està definida com un *TabularInline* per a que l'usuari pugui afegir imatges amb el *extra=1* fins a un màxim de 5, amb el *max_num = 5*.

4.2.3.- CLASSE ADMINCONCURSANT

```
class AdminConcursant(admin.ModelAdmin):
    # Definim els caps a mostrar en la part administrativa pel que fa al concursant
    list_display = ('nom', 'cognom1', 'cognom2', 'correu')
    list_filter    = ('cognom1',)
    ordering = ('cognom1',)
    search_fields  = ('nom', 'cognom1', 'cognom2',)
    inlines = [ImatgeInline,]

admin.site.register(Concursant, AdminConcursant)
```

Imatge 51: admin3

La classe *AdminConcursant* és la classe que definim per poder administrar els concursants des de la part administrativa. Ens permetrà afegir, eliminar o modificar tot allò que fa referència al concursant.

- ***list_display*** l'utilitzem per dir-li quins camps volem que es mostrin en obrir *Concursants* des de la part administrativa.
- ***list_filter*** ens serveix per filtrar. En el nostre cas pel *cognom1*.
- ***ordering*** el que fa es ordenar pel camp que li diem, *cognom1*.
- ***search_fields*** l'utilitzem per buscar un concursant en concret i li hem de dir per quins camps ha de buscar. Permetem fer una cerca per *nom*, *cognom1* i *cognom2*.

Per relacionar-ho amb les imatges utilitzem *inlines = [ImatgeInline,]* així des del propi concursant ja podem afegir imatges fins a un màxim de 5 com s'ha definit anteriorment.

Per guardar-ho tot, utilitzem *admin.site.register(Concursant, AdminConcursant).*

4.2.4.- CLASSE ADMINIMATGES

```
class AdminImatges(admin.ModelAdmin):
    # Definim els caps a mostrar en la part administrativa pel que fa a la imatge
    fieldsets = (
        (None, {'fields': ('nom_imatge', 'imatge')}),
        )
    list_display = ('concursant', 'nom_imatge', 'imatge', 'creada', 'modificada',)
    list_filter    = ('nom_imatge',)
    search_fields  = ('nom_imatge',)

admin.site.register(Imatge, AdminImatges)
```

Imatge 52: admin4

La classe *AdminImatges* té un camp diferent de *AdminConcursant*. Aquest camp és el *fieldsets*.

- ***fieldsets*** ens serveix per a que apareguin els camps *nom de la imatge* i la opció *d'adjuntar la imatge* a l'hora d'afegir o modificar.
- ***list_display*** l'utilitzem per dir-li quins camps volem que es mostrin en obrir *Imatge* des de la part administrativa.
- ***list_filter*** ens serveix per filtrar, en el nostre cas pel *nom_imatge*.
- ***search_fields*** l'utilitzem per buscar una imatge en concret, fent la cerca per *nom de la imatge*.

Per guardar-ho tot, utilitzem *admin.site.register(Imatge, AdminImatge).*

4.2.5.- CLASSE ADMINJURAT

```
class AdminJurat(admin.ModelAdmin):
    # Definim els caps a mostrar en la part administrativa pel que fa al jurat
    list_display = ('nom',)
    search_fields  = ('nom',)
    inlines = [PuntsimatgeInline,]

admin.site.register(Jurat, AdminJurat)
```

Imatge 53: admin5

Per finalitzar amb el codi de la part administrativa, l'última classe que podem observar és la classe *AdminJurat*. Aquesta classe consta de:

- ***list_display*** que ens mostrarà en entrar a *Jurat* els membres que formen el jurat.
- ***search_fields*** que ens permetrà buscar pel *nom* del membre del jurat.

Per relacionar-ho amb les imatges utilitzem *inlines = [PuntsimatgeInline,]* així des del propi *jurat* ja podem ficar les notes a les imatges que escollim.

4.2.6.- ADMIN.PY VISUALMENT

Després de crear el codi que pertany a la part administrativa ja podem accedir-hi per veure el resultat. Aquesta part ens permetrà afegir i modificar el contingut al nostre gust.

Per al nostre servidor hem d'escriure la ruta http://fotomath/admin en el navegador i ens apareixerà la imatge següent:

Imatge 54: admin6

Aquí hi escriurem *l'usuari* i la *contrasenya* que hem introduït al donar d'alta la base de dades tal com ja s'ha descrit anteriorment. Si introduïm les dades correctament ens portarà a una pantalla similar a la de la imatge següent:

Imatge 55: admin7

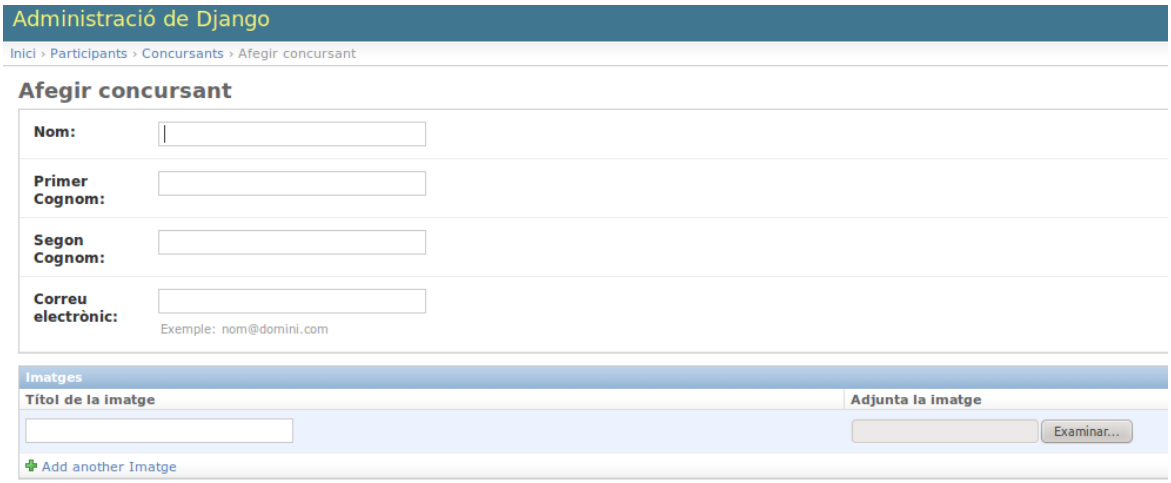
Aquests camps que podem observar són aquells que hem creat a la part administrativa. En aquesta part ja podem administrar el nostre lloc d'una manera senzilla, modificar o afegir concursants, imatges... veure llistats dels participants, ordenar-los, filtrar o buscar... tot el que hem creat per a l'aplicació *participants*.

Si es prem sobre Concursants se'ns mostrarà una imatge com la següent.

Imatge 56: Admin Concursants

En aquesta imatge podem observar el resultat del codi realitzat anteriorment dins la classe *AdminConcursant*.

Si prenem a afegir al marge dret ens apareixerà una imatge com la que es mostra a continuació, on podrem donar d'alta concursants. Si seleccionem un concursant podrem modificar-ne el contingut o fins i tot eliminar-lo.



Imatge 57: Afegir Concursant

L'entrada de dades és la definida des del *Model*, on hem definit unes característiques per a cada camp i s'espera rebre les dades tal com s'han definit. Si falta algun camp per omplir no ens deixaria crear l'usuari. Des d'aquest mateix quadre de diàleg podem adjuntar les imatges fins a un màxim de 5.

Dins l'apartat *d'imatges*, el funcionament és exactament igual que el de *concursant* pel que fa a afegir, modificar o eliminar imatges. En la imatge següent podem observar una imatge pujada des de concursant, amb el nom del concursant complert, el títol de la imatge, la imatge pujada i la data de creació i última modificació.

Imatge 58: Admin Imatges

Si es vol afegir una imatge més només cal pitjar a afegir i s'obra un nou quadre de diàleg amb una imatge semblant a la que es mostra a continuació on ens demana el títol de la imatge i un altre camp per poder pujar-la.

S'observa que tots els camps tenen un nom "amigable" i no el valor de la variable utilitzada. Això ho hem fet al *model* amb els *verbose_name*.

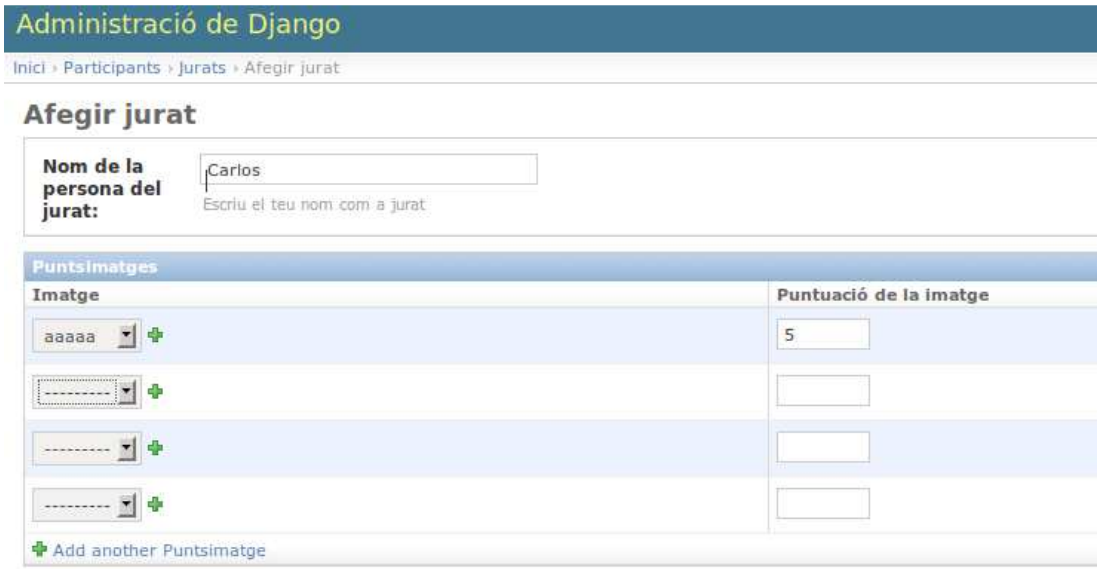
Tot el que s'ha desenvolupat, ni que siguin fitxers diferents, estan lligats gràcies als *imports* que s'han realitzat. Per aquest motiu es important realitzar primer el model (la nostra base de dades i estructura) i després anar desenvolupant els diferents fitxers lligant-los amb els *imports* per importar el contingut d'un lloc a un altre.

Imatge 59: Afegir Imatges

El tercer punt que hi ha a la part administrativa, a l'apartat dels participants, és el del jurat. Des d'aquest apartat també podrem donar d'alta, modificar o eliminar membres del jurat.

Imatge 60: Admin Jurat

Si prenem a afegir, donarem d'alta a un membre del jurat i des de la mateixa pàgina aquest membre podria valorar totes les imatges que estiguessin ja a la base de dades, s'ha cregut convenient realitzar-ho d'aquesta manera, per facilitar la tasca dels membres del jurat a l'hora de puntuar les imatges. Una mostra de com seria el quadre de diàleg, ho podem observar a la imatge següent:



Imatge 61: Afegir Jurat

Si hi ha hagut algun error en la valoració, es podria modificar sense cap tipus de problema.

Dins l'últim apartat dels participants observem el camp punts imatge, on es mostren totes les notes de les imatges de la base de dades. A l'hora d'escollir les imatges amb més nota és fàcil buscar quines han estat aquestes.

Imatge 62: Admin Punts Imatge

Per valorar les imatges, els membres del jurat també ho poden fer des d'aquest apartat, prement a afegir, es mostra un quadre de diàleg com la següent:

Imatge 63: Afegir Punts Imatge

On el jurat pot escollir la imatge a valorar, quina persona del jurat realitza la valoració i la nota que li dona a la imatge.

L'acció de modificar el contingut és igual de simple que la d'afegir. En la imatge següent es mostra una imatge del que apareix al donar-li a modificar.

Imatge 64: Modificar Punts Imatge

Des del menú desplegable buscaríem la imatge a modificar. Els altres dos camps s'omplen automàticament amb el contingut antic i només caldria modificar el contingut desitjat.

Arribats a aquest punt ja tenim creada la nostra base de dades i una part administrativa per poder administrar el lloc. Anem ara a crear el formulari de registre de concursants i enviament de les imatges. Per fer-ho crearem un arxiu nou anomenar forms.py.

4.3.- FORMS.PY

L'arxiu *forms.py* el crearem dins de la nostra aplicació *participants.* Aquest arxiu serà l'encarregat de mostrar el formulari i anirà lligat amb l'arxiu *models* de la nostra base de dades.

```
# -*- coding: utf-8 -*-
from django import forms
from participants.models import Concursant, Imatge
from captcha.fields import CaptchaField
from django.forms.models import BaseInlineFormSet, inlineformset_factory


class ConcursantForm(forms.ModelForm):
    # Definim el formalari concursant
    class Meta:
        model = Concursant
        fields = ('nom', 'cognom1', 'cognom2', 'correu')
```

Imatge 65: Formulari part1

El primer que hem de fer es importar dins el codi de l'arxiu forms.py tot el contingut que hem d'utilitzar. Això ho fem amb els *imports* del principi de l'arxiu.

Des de *Django*, importem els *forms* (formularis) per poder utilitzar les definicions i comandes dels formularis de *Django*.

També importem des de l'aplicació *participants* el nostre *models*, però només allò que hem d'utilitzar: *Concursant* i *Imatge* que són les classes on hem definit anteriorment aquets dos camps.

El tercer *import* que podem observar és el que fa referencia als *captcha*, una mesura de seguretat per evitar enviaments massius. Des dels arxius de *captcha* importem el *CaptchaField.*

Per finalitzar amb els *imports* també importem des dels arxius de *Django* dins de *forms* i dins de *models*, el *BaseInlineFormSet* i *inlineformset_factory* ja que els necessitarem per poder definir el codi.

Després de les importacions ja podem començar a definir el codi del formulari. Comencem definint la classe *ConcursantForm*. S'escull aquest nom perquè farà referencia al concursant dins del formulari.

Per relacionar la classe creada al formulari amb la creada al model, creem una subclasse dins de *ConcursantForm*, anomenada *Meta*, on li diem que el *model* al que fa referència és el de la classe *Concursant* de la nostra base de dades i que els camps que volem que ens mostri són els camps *nom, cognom1, cognom2 i correu*. Això ho fem amb el *fields*. Així el participant podrà omplir aquests camps en el formulari i es guardaran a la base de dades ja que li hem fet la relació.

Ja tenim definida la part de les dades dels participants, anem ara a definir la part pertanyent a les imatges que aquets faran arribar per participar al concurs.

Per definir la part del formulari pertinent a les imatges el codi no és tant simple com ho hem definit pel que fa al *Concursant*. Hem de crear una validació dels camps i un control sobre les imatges dels participants.

Primer definim una classe per a portar el control respecte a si s'han afegit imatges o informar a l'usuari que almenys han d'adjuntar una imatge. Per fer-ho definim la classe *ImatgeBaseInlineFormSet* que tal com s'ha dit serà l'encarregada de fer les validacions dels camps.

A continuació es mostra una imatge de la classe creada pel control de les imatges:

```
class ImatgeBaseInlineFormSet(BaseInlineFormSet):
    # Definim el formulari imatge base per poder fer validacions dels camps
    def clean(self):
        """ Obté els formularis que tenen dades correctes"""
        super(ImatgeBaseInlineFormSet, self).clean()
        contador = 0
        for form in self.forms:
            try:
                if form.cleaned_data:
                    contador += 1
            except AttributeError:
                # Si un dels subformularis no és correcte,
                # informa de AttributeError per al cleaned_data
                pass
        if contador < 1:
            raise forms.ValidationError("Heu d'enviar almenys una fotografia")
```

Imatge 66: Formulari part2

En desenvolupar el codi s'han anat ficant anotacions per a que sigui més entenedor. Aquesta porció de codi comprova que en el camp de la imatge hi hagi adjuntada una imatge. Si el participant no ha adjuntat cap imatge retorna un error de validació que l'informa que almenys ha d'enviar una fotografia. Aquesta comprovació es fa utilitzant un comptador. Es comprova que s'ha adjuntat una imatge. Si és així el comptador, es fica a 1 en cas contrari salta el error.

Ara que ja tenim definit el formulari base d'imatge anem a definir les limitacions del camp imatge per a portar un control sobre les imatges que ens fan arribar. Per fer-ho creem una nova classe on definirem la mida màxima (el pes) de la imatge i els formats d'enviament admesos. Si les imatges rebudes no compleixen aquests requisits seran rebutjades i s'informarà al participant de l'error generat.

En la imatge que es mostrarà a continuació (Imatge 66: Formulari part3) definim la classe *ImatgeForm* amb les limitacions anteriorment esmenades. Per relacionar aquesta classe amb la base de dades (*model*) fem el mateix que amb el *Concursant*. Creem una subclasse anomenada *Meta* on li assignem el *model = Imatge*. Volem que ens mostri els camps nom de la imatge i un camp imatge per poder adjuntar-la.

```
class ImatgeForm(forms.ModelForm):
    # Definim el formulari imatge
    class Meta:
        model = Imatge
        fields = ('nom_imatge', 'imatge')

    def clean_imatge(self):
        """ Rebutja les imatges > 3MB i les que no estan en format jpeg """
        max_mida = 3*2**20
        formats_imatge = ['image/jpeg']
        imatge = self.cleaned_data['imatge']
        tipus_imatge = imatge.content_type
        if tipus_imatge in formats_imatge:
            if imatge._size > max_mida:
                raise forms.ValidationError(
                    'La imatge ha de ser menor que %d bytes.' % max_mida
                    )
        else:
            raise forms.ValidationError("La imatge ha d'estar en format jpeg")
        return imatge
```

Imatge 67: Formulari part3

Les limitacions pel que fa a la mida de la imatge i els formats admesos, ho definim dins del *def clean_image(self).*

Primer definim la mida màxima de la imatge (3 MB) i després el format de la imatge (jpeg). Es comprova si el format de la imatge es correcte. De ser-ho, comprova la mida de la imatge. Si aquesta és més gran que el *max_mida* retorna un error amb la mida de la imatge correcta. Si pel contrari la mida de la imatge és correcta però el format no, retorna un error informant amb el format correcte de les imatges.

Ara ja tenim ben definits els camps concursant i imatge del formulari però volem que el participant, si ho vol, pugui enviar més d'una imatge. Per fer-ho li hem de dir:

```
ImatgeFormSet = inlineformset_factory(
    Concursant,
    Imatge,
    form = ImatgeForm,
    formset = ImatgeBaseInlineFormSet,
    can_delete=False,
    extra=5)
```

Imatge 68: Formulari part4

El màxim d'imatges possibles a adjuntar ho definim amb el *extra=5*.

Ja tenim definit el nostre formulari. Anem ara a definir una mesura de seguretat per evitar enviaments massius, el *captcha*.

El *captcha* és una imatge i una caixa de text on el participant al concurs ha d'escriure a la caixa de text el que pot observar a la imatge. La imatge i el que escriu han de coincidir, de no ser així no valida el formulari.

```
class CaptchaTestModelForm(forms.ModelForm):
    # Definim el formulari captcha per poder fer validacions del camp
    captcha = CaptchaField()
    class Meta:
        model = Concursant
        exclude = ('nom', 'cognom1', 'cognom2', 'correu',)
```

Imatge 69: Formulari part5

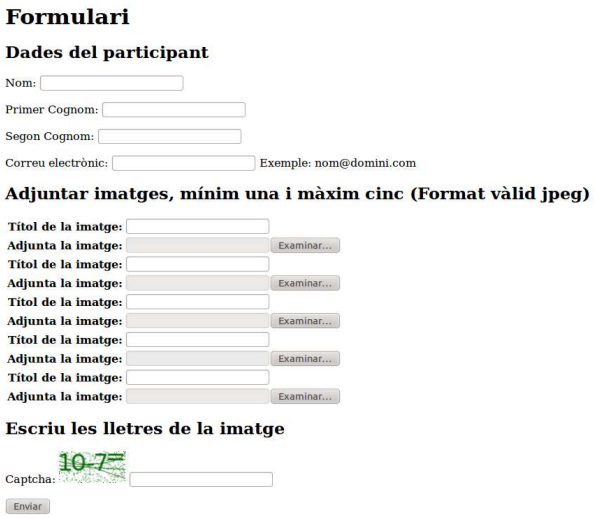
El *captcha* com les *imatges* i el *concursant* s'han de vincular al *model*. Per fer-ho fem com en el cas del *Concursant* i de les *Imatges* amb la subclasse *Meta*, però en aquest cas no volem que ens mostri els camps un altre cop, només volem que ens mostri el *captcha*. Així, el que fem es excloure tots els camps del *Concursant*.

Ja tenim definida l'estructura i confeccionat el codi del nostre formulari. Anem a veure una mica com funciona i veure si el control d'errors implementat funciona correctament.

4.3.1.- EL FORMULARI

En la imatge següent es mostra com és el formulari en la màquina de proves. No se li ha donat format amb els *CSS* ja que per la màquina de proves no es necessari. Per aquest motiu apareix només amb blanc i negre i sense un format específic.

Per poder observar aquesta plana haurem de definir un *template* que ens mostri el contingut del formulari, això ho explicarem en l'apartat de *templates*.



Imatge 70: Formulari

Si el participant ha emplenat totes les dades correctament li apareixerà una nova pàgina informant-lo que les seves dades han estat enviades correctament.

Felicitats! les seves dades han estat enviades correctament

Imatge 71: Imatge Correcta

Igual com s'ha esmentat en el formulari, per a que es mostri aquesta pàgina amb el missatge s'ha de crear un *template* per aquesta funció. En parlarem en l'apartat corresponent.

Ara ja tenim definit el formulari i hem vist com funciona. Per a que el formulari funcioni, no només ha d'anar lligat al *model.py*, sinó que també ha d'anar lligat al *views.py*, al *urls.py* i als *templates*.

Per a que el formulari guardi les entrades de dades del concursant i les seves imatges hi ha d'haver : el *formulari* que digui quins camps hi ha d'haver, un *template* que mostri el formulari, una *vista* que ho interpreti i guardi a la base de dades i una *base de dades* on guardar-ho.

Dit això anem ara a veure les vistes que s'han fet per al bon funcionament del nostre formulari.

4.4.- VIEWS.PY

Com ja hem descrit en el punts anteriors, el primer que hem de fer es importar tot allò que necessitarem per escriure el codi i que realitzi les tasques que nosaltres desitgem.

Les vistes (views) són les responsables de:

- ***Tornar un objecte HttpResponse amb el contingut de la pàgina sol·licitada,***
- ***O bé, llençar una excepció com Http404.***

Generalment, una vista recupera dades d'acord als paràmetres, carrega una plantilla i l'omple amb les dades recuperades.

La primera funció que trobem i la principal, es l'encarregada de gestionar el formulari. Va molt lligada amb el nostre *forms.py* que hem descrit en el capítol anterior.

La primera línia és la definició, el nom que li assignem a la funció. En la segona línia el que podem observar és què el codi fa una comprovació de si el formulari ha estat enviat i si ha estat amb el mètode *POST*. Una altra manera de fer-ho hagués pogut ser amb *GET*, però per als nostres objectius el millor mètode era *POST*, ja que aquest mètode permet enviar les dades dels participants amb més seguretat.

En les línies següents comprovarem que s'ha rebut el formulari pertinent al concursant. Mirarem que aquestes dades hagin estat enviades correctament. Després farem el mateix pel formulari de les imatges i el formulari del *captcha* i si tots tres han estat enviats correctament guardarem el contingut i retornarem al participant una nova pàgina on li donarem les gràcies. En cas contrari, retornarem a l'usuari a la part on no s'han introduït les dades correctament.

```
# -*- coding: utf-8 -*-
from django import forms
from django.template import TemplateDoesNotExist, RequestContext
from captcha.fields import CaptchaField
from django.shortcuts import get_object_or_404, render_to_response
from django.http import HttpResponse, Http404, HttpResponseRedirect
from django.conf import settings
from participants.forms import ConcursantForm, ImatgeFormSet, CaptchaTestModelForm
from participants.models import Concursant, Imatge
import unicodedata
from django.core.urlresolvers import reverse


def gestio_concursants(request):
    # Si el formulari ha estat enviat:
    if request.method == 'POST':
        # El formulari del concursant emplenat:
        concursant_form = ConcursantForm(request.POST)
        captcha_form = CaptchaTestModelForm(request.POST)
        # Validació del formulari concursant:
        if concursant_form.is_valid():
            concursant = concursant_form.save(commit=False)
            imatge_formset = ImatgeFormSet(request.POST, request.FILES, instance=concursant)
            # Validació del formulari foto i del capcha:
            if imatge_formset.is_valid() and captcha_form.is_valid():
                concursant.save()
                imatge_formset.save()
                captcha_form.save()
                human = True

                return HttpResponseRedirect('/gracies/')

        else:
            imatge_formset = ImatgeFormSet(instance=Concursant())

    else:
        concursant_form = ConcursantForm()
        imatge_formset = ImatgeFormSet(instance=Concursant())
        captcha_form = CaptchaTestModelForm()

    return render_to_response('formulari.html', {
        'MEDIA_URL': settings.MEDIA_URL,
        'concursant_form': concursant_form,
        'imatge_formset': imatge_formset,
        'captcha_form': captcha_form},
        context_instance=RequestContext(request))
```

Imatge 72: Views.py Formulari

Com ja s'ha comentat, si l'usuari envia les dades correctament, li retornem una nova pàgina on li donem les gràcies. Per poder mostrar plana hem de fer una vista que faci aquesta funció, *def gracies*, l'únic que fa es retornar una plana web amb el missatge de gràcies que va a buscar al *template* on està creada.

```
def gracies(request):
    pag = "gracies.html"
    return render_to_response(pag)
```

Imatge 73: Views.py Formulari segona part

Després de definir les nostres *vistes*, haurem de definir les *urls*.

4.5.- URLS.PY

Les *url* són les encarregades de mostrar les pàgines web. Així, cada funció està assignada amb una plana web, realitzada als *templates*. Aquesta és la seva tasca: vincular les *vistes* amb els *templates*.

En el patró de la url s'utilitzen parèntesis per capturar una part de la mateixa i poder accedir-hi després com a paràmetres a la vista. L'expressió entre parèntesis es denomina grup i són pròpies de les expressions regulars en Python.

```
# -*- coding: utf-8 -*-
from django.conf.urls.defaults import *
from views import gestio_concursants, gracies
from django.contrib import admin
from settings import MEDIA_ROOT
admin.autodiscover()

urlpatterns = patterns('',
    url(r'^captcha/', include('captcha.urls')),
    (r'^formulari/', gestio_concursants),
    (r'^gracies/', gracies),
    (r'^admin/', include(admin.site.urls)),
)
```

Imatge 74: *Urls.py*

Tal com s'ha fet en tots els casos, el primer serà importar tot el contingut que puguem necessitar.

Després de les importacions ja podem declarar les *url*.

Totes les declaracions d'*url* es fan dins del *urlpatterns*. En aquesta mena de taula és on assignem o li diem on ha d'anar a buscar la plana.

En la primera declaració:

```
url(r'^captcha/', include('captcha.urls'))
```

Li estem dient que el *captcha* està inclòs dins del *captcha.url*. És allí on ha d'anar a buscar el contingut a mostrar.

Pel que fa a la declaració del *formulari*, li estem dient que ha d'anar a buscar la funció definida a *gestió_consursants* dins la nostra *vista* i mostrar el *template* del *formulari*. Aquests passos es realitzen així successivament per totes les importacions que hem creat. La primera part fa la crida i executa la segona part, aquesta pot ser una pàgina web, una funció, ...

```
url(r'^gracies/', gracies),
```

En aquesta declaració li estem dient que si es crida gracies dins de la funció ha de mostrar/executar la pàgina de gracies que tenim definida.

4.6.- TEMPLATES (PLANTILLES)

Els *templates* (plantilles) són pàgines *HTML*. *S*eran les encarregades de representar tot el codi generat anteriorment en una pàgina web.

No és una bona idea codificar *l'html* directament en les *vistes*. És molt més net i més fàcil de mantenir separat el disseny de la pàgina del codi *Python*. Aquest procés és realitza a través del sistema de plantilles de *Django*.

Una plantilla de Django és una cadena de text que separa la presentació d'un document de les seves dades. Per tant, defineix contenidors i diversos bits de lògica bàsica (etiquetes) que regulen la forma en què el document ha de ser mostrat. En general les plantilles s'utilitzen per produir html, però concretament les de Django són igualment capaces de generar qualsevol altre format basat en text.

Django utilitza el concepte de "*herència de plantilles*":

```
{% extends "base.html"%}
```

Significa "*Primer carrega la plantilla anomenada 'base', la qual té alguns blocs definits, i omple'ls amb els següents blocs*".

En altres paraules, permet disminuir dràsticament la redundància en les plantilles. Així, cada plantilla ha de definir només el que li és propi.

En la imatge que es mostrarà a continuació encara no s'ha afegit aquesta part però serà necessària per a lligar-la amb l'aplicació principal. També s'hi pot observar el *template* encarregat de representar el formulari de registre i enviament d'imatges per als participants al concurs.

```
<html>

<title>Formulari entrada dades</title>

<body>

<h1>Formulari</h1>

<h2>Dades del participant</h2>

<form enctype='multipart/form-data'  action='.' method='post'>{% csrf_token %}
    {{ concursant_form.as_p }}
    <table>
    <h2>Adjuntar imatges, mínim una i màxim cinc (Format vàlid jpeg)</h2>
    {{ imatge_formset.management_form }}
    {{ imatge_formset.non_form_errors }}
    {{ imatge_formset.as_table }}
    </table>
    <h2>Escriu les lletres de la imatge</h2>
    {{ captcha_form.as_p }}

    <p><input type="submit" value="Enviar" /></p>
</form>

</body>

</html>
```

Imatge 75: Template - Formulari.html

Aquest codi mostrarà un formulari en columna on hi consten per a cada concursant totes les seves imatges. El format, doncs, bé definit pel *form* que mostra un conjunt de formularis agrupats: el del concursant, el de les imatges i el del *captcha*.

Tot seguit podem observar l'*imatge*_formset, on mostrarà uns requadres on el participant podrà prémer per adjuntar la imatge, ficar el títol... i fora d'aquesta taula mostrarà el *captcha*. El text d'ajuda el mostrem utilitzant els *h2*.

A més ha d'enviar el formulari omplert un cop realitzada l'acció de prémer el botó d'enviament i ho ha de fer amb mètode POST.

Crearem un altre *template* que serà l'encarregat de mostrar una nova pàgina amb el missatge *"Felicitats! Les seves dades han estat enviades correctament"*. Cada pàgina web que vulguem mostrar ha de tenir un *template* que la mostri o la representi. (Aquesta plana s'ha afegit per realitzar proves de funcionament)

```
<html>
<title>Dades introduides correctament</title>
<body>
<h1>Felicitats! les seves dades han estat enviades correctament</h1>
</body>
</html>
</form>
```

Imatge 76: Template - Gràcies.html

En aquesta part, com ja s'ha fet amb l'aplicació del formulari web, es comentarà el sistema utilitzat per mostrar les imatges dins la web del servidor de proves i s'analitzaran els canvis, de modificació o creació, que han patit els arxius un cop s'han implementat en el servidor definitiu.

5.- APLICACIÓ 2: GESTIÓ D'IMATGES

La segona aplicació a desenvolupar per aquest projecte és el sistema encarregat de gestionar i mostrar les imatges que els participants faran arribar al concurs mitjançant el formulari de registre i enviament de dades i imatges.

Primerament aquesta aplicació es pretenia desenvolupar en *Django*. Per fer-ho es va consultar molta informació sobre el tractament d'imatges en *Django* i, tot i el seu potencial en la resta de mòduls, els resultat de les cerques mostraven que l'òptim pels diferent autors era *JavaScript*.

Django és un llenguatge molt nou amb la virtut que se li pot afegir o lligar altres llenguatges de programació. Gràcies al seu funcionament d'aquests llenguatges es pot programar aplicacions de manera senzilla i potent per després lligar-les amb *Django* per a que funcionin com a una sola aplicació.

El que s'ha fet ha estat adaptar un *codi obert de JavaScript*[6] que realitzava unes funcions similars a les que nosaltres volíem desenvolupar. Després de fer les adaptacions pertinents només calia lligar-ho amb la nostra aplicació.

Per adaptar aquest el nou codi a les nostres necessitats s'han afegit línies als arxius creats prèviament per al formulari que s'encarreguen de gestionar les imatges. Concretament, els arxius modificats són *view*, *url* i s'ha creat un *template*.

Les dues imatges que es mostren en els subapartats següents fan referència a dos funcions que s'han afegit als arxiu *views.py*.

5.1.- VIEWS.PY

La definició *mostrar_imatge* l'hem afegit a continuació de la del *concursant*.

En aquesta definim com es mostraran el conjunt de les imatges enviades fent una consulta a la base de dades i carregant-les des de el directori on estan guardades. A part de la imatge en miniatura, en fer-se gran la imatge ha de mostrar el seu nom i el del seu autor.

[6] *El codi de JavaScript s'ha afegit a la documentació del CD ja que és un codi extens de 3300 línies.*

```
def mostrar_imatges(request):
    total_imatges = Imatge.objects.all()
    if len(total_imatges) >0:
        imatge0 = total_imatges[0]
        nom_directori_diapositives = '.diapositives'
        #
        url_imatges = os.path.split(imatge0.imatge.url)[0]
        url_diapositives = os.path.join(url_imatges, nom_directori_diapositives)
        #
        directori_imatges = os.path.split(imatge0.imatge.path)[0]
        directori_diapositives = os.path.join(directori_imatges, nom_directori_diapositives)
        try:
            os.mkdir(directori_diapositives, 0755)
        except os.error, msg:
            pass
        #
        llista_diapositives = [f for f in os.listdir(directori_diapositives)
                               if os.path.isfile(os.path.join(directori_diapositives, f))]
        #
        dades_imatges = []
        for f in total_imatges:
            if os.path.isfile(f.imatge.path):
                nom_diapositiva = 'dp_' + os.path.split(f.imatge.path)[1]
                diapositiva = os.path.join(url_diapositives, nom_diapositiva)
                #
                if nom_diapositiva not in llista_diapositives:
                    fer_diapositiva(f.imatge)
                #
                foto = f.imatge.url
                titol = f.titol
                autor = '' #
                dades = {'autor':autor, 'foto':foto, 'titol':titol, 'diapositiva':diapositiva}
                dades_imatges.append(dades)

    return render_to_response('expo.html', {
            'MEDIA_URL': settings.MEDIA_URL,
            'dades_imatges': dades_imatges,
            }, context_instance=RequestContext(request))
```

Imatge 77: Views gestió d'imatges

En la següent imatge, definim una nova funció.

```
def fer_diapositiva(fitxer, mida='104x104'):
    f = fitxer.path
    x, y = [int(x) for x in mida.split('x')]
    #
    cami_a_fitxer, nom_fitxer = os.path.split(f)
    nom_diapositiva = 'dp_' + nom_fitxer
    diapositiva = os.path.join(cami_a_fitxer, '.diapositives', nom_diapositiva)
    #
    imatge = Image.open(f)
    imatge.thumbnail([x, y], Image.ANTIALIAS)
    #
    try:
        imatge.save(diapositiva, imatge.format, quality=90, optimize=1)
    except:
        imatge.save(diapositiva, imatge.format, quality=90)
```

Imatge 78: Views gestió d'imatges segona part

Aquesta funció aconsegueix disminuir la mida de les imatges facilitades per l'autor de tal manera que en una vista prèvia es puguin mostrar totes les imatges participants al concurs i, si es desitja, es pugui observar la imatge seleccionada en la mida original.

5.2.- TEMPLATES

Per poder mostrar les imatges hem hagut de crear un nou *template*. En aquest s'ha afegit el *extends* "plantilla.html" per indicar que és una pàgina web lligada a la principal i per tant és una extensió.

El codi corresponent al *script* és el que ens lliga el nostre *views* amb l'arxiu *JavaScript* . Aquest arxiu, l'hem carregat a la carpeta *js* dins de *sitemedia*. És el *template* l'encarregat de mostrar les imatges tal com ho hem definit en el *views*.

```
{% extends "plantilla.html" %}

{% block headexpo %}
<script type="text/javascript" src="{{ MEDIA_URL }}js/highslide-full.js"></script>
<link rel="stylesheet" type="text/css" href="{{ MEDIA_URL }}css/highslide.css" />
<!--[if lt IE 7]>
<link rel="stylesheet" type="text/css" href="{{ MEDIA_URL }}css/highslide-ie6.css" />
<![endif]-->
<script type="text/javascript">
        hs.cssDirection = 'ltr';
        hs.loadingText = 'Carregant...';
        hs.loadingTitle = 'Feu clic per cancel·lar';
        hs.focusTitle = 'Feu clic per portar al capdavant';
        hs.fullExpandTitle = 'Ampliar a mida real (f)';
        hs.creditsText = 'Desenvolupat per <i>Highslide JS</i>';
        hs.creditsTitle = 'Anar a la pàgina principal de Highslide JS';
        hs.previousText = 'Anterior';
        hs.nextText = 'Següent';
        hs.moveText = 'Moure';
        hs.closeText = 'Tancar';
        hs.closeTitle = 'Tancar (esc)';
        hs.resizeTitle = 'Canviar la mida';
        hs.playText = 'Reproducció';
        hs.playTitle = 'Reproducció de diapositives (barra espaiadora)';
        hs.pauseText = 'Pausa',
        hs.pauseTitle = 'Pausa de diapositives (barra espaiadora)';
        hs.previousTitle = 'Anterior (fletxa esquerra)';
        hs.nextTitle = 'Següent (fletxa dreta)';
        hs.moveTitle = 'Moure';
        hs.fullExpandText = '1:1';
        hs.number = '';
        hs.restoreTitle = 'Feu clic per tancar la imatge, feu clic i arrossegueu per moure. Utilitzeu les tecles de fletxa per següent i anterior.';

  hs.graphicsDir = '{{ MEDIA_URL }}images/highslide/';
  hs.align = 'center';
  hs.transitions = ['expand', 'crossfade'];
  hs.fadeInOut = true;
  hs.dimmingOpacity = 0.8;
  hs.outlineType = 'rounded-white';
  hs.captionEval = 'this.thumb.alt';
  hs.marginBottom = 105;
  hs.numberPosition = 'caption';
  hs.showCredits = false;

  hs.addSlideshow({
    interval: 5000,
    repeat: false,
    useControls: true,
    overlayOptions: {
      className: 'text-controls',
      position: 'bottom center',
      relativeTo: 'viewport',
      offsetY: -60
    },
    thumbstrip: {
      position: 'bottom center',
      mode: 'horizontal',
      relativeTo: 'viewport'
    }
  });
</script>
{% endblock %}
```

Imatge 79: Template – Expo.html

La imatge l'hem dividit en dues parts per poder mostrar-la millor. En aquesta segona part definim la mida que tindran les imatges dels participants i com s'han de mostrar quan siguin ampliades.

```
{% block expo %}current{% endblock %}

{% block contingut %}
<h2>Exposici&oacute;</h2>
<p>Fotograf&iacute;es participants en aquest cert&agrave;men.</p>
<div class="highslide-gallery" style="width: 600px; margin: auto">
{% for dada in dades_imatges %}
        <a class='highslide' href='{{ dada.foto }}' onclick="return hs.expand(this)">
                <img src='{{ dada.diapositiva }}' alt='{{ dada.titol }}'/></a>
{% endfor %}
</div>
{% endblock %}
```

Imatge 80: Template – Expo.html segona part

5.3.- URLS.PY

Pel que fa al fitxer *urls.py*, s'afegeix en les primers línies a *l'import* de *gestionar_concursant* un atribut més per a que importés també *mostrar_imat*ges.

Després de fer les importacions s'ha afegit la línia (*r'^expo$', mostrar_imatges*).

```
    (r'^(inici|bases|premis|entrega|jurat|lligams)$', the_page),
    (r'^formulari$', gestionar_concursant),
    (r'^expo$', mostrar_imatges),
    #
    url(r'^captcha/', include('captcha.urls')),
    #
    (r'^admin/', include(admin.site.urls)),
)
```

Imatge 81: URL imatge

6.- SERVIDOR FOTOMATH

Després de modificacions i proves es dona per finalitzada la tasca de desenvolupar les aplicacions necessàries per a la II edició del Concurs FotoMath:

- el formulari web, que gestiona els concursants i
- la part de gestió de les imatges dels participants, que permet mostrar-les en diferents mides i que el jurat pugui avaluar-les.

La següent tasca és la implantació en el servidor de treball real. Com *Django* funciona amb mòduls només cal afegir-hi aquells que nosaltres hem creat.

Com a mesura preventiva implementa un sistema de còpies de seguretat de la base de dades. Aquest punt no és va tenir en compte en el servidor de proves ja que les dades introduïdes eren irrellevants.

Un cop fetes les proves de control d'errors al servidor definitiu s'obren les dos aplicacions als usuaris per a que poguessin començar a enviar les imatges participants al concurs.

7.- INCOMPATIBILITATS I PROBLEMES

En tot desenvolupament d'aplicacions apareixen incompatibilitats o problemes, ja siguin pel tipus de programari utilitzat o pel d'elaboració pròpia.

El principal problema que s'ha tingut ha estat la **falta d'informació** sobre *Django*. És cert que per Internet hi ha gran quantitat d'enllaços que en parlen però tota la documentació i els exemples són sempre redundants. Majoritàriament tota la informació disponible a la xarxa fa referència a la web oficial de *Django*.

Aquest fet ha generat importants inconvenients a l'hora d'elaborar les nostres pròpies aplicacions ja que no existeix cap exemple concret que s'adeqüi a les nostres característiques. Tots els exemples que poden estar relacionats d'alguna manera són, a data d'avui, bastant senzills.

Un altre problema que no s'havia tingut en compte des del principi van ser les **actualitzacions de sistema** que realitza *Linux* periòdicament. Aquestes afecten tant el sistema operatiu base com tot el programari que s'hi ha instal·lat. Així, cada cop que es realitzava una actualització hi havia el perill de perdre la configuració aplicada al programa i que afectava al projecte. Un cop detectada la situació es van desactivar les actualitzacions automàtiques mentre l'aplicació estava en fase de desenvolupament com a últim recurs ja que no hi ha documentació de com pal·liar el problema.

Concretament, quan es realitzava l'actualització de *PostgreSQL*, de la 8.4 a la 9.0 i després a la 9.1, s'havia de modificar la configuració del *Settings* i canviar el port de la base de dades. De no ser així, el sistema no s'havia on havia d'anar a buscar la base de dades.

En Linux, el tema de **permisos** és important ja que si una aplicació no té els permisos per accedir a una informació, sorgeixen problemes en l'execució. En el nostre cas aquest fet ens va aparèixer a l'hora de realitzar la càrrega d'imatges des de el nostre *model*. Es va haver de modificar els permisos a les carpetes necessàries per a que la càrrega d'imatges fos efectiva.

Un altre error que vam observar a l'hora de **tractar les imatges**, era que en realitzar la càrrega amb el *upload* de *ImageField* s'havia d'especificar la ruta relativa al nostre *setting MEDIA_ROOT* en la que volíem que es guardés l'arxiu pujat mitjançant l'argument *upload_to*.

Si utilitzem una barra / al començament de la ruta en *upload_to* l'arxiu ja no es puja a la ruta especificada de forma relativa a *MEDIA_ROOT* sinó que s'intenta guardar en aquesta ruta a partir del directori arrel.

Per exemple, amb `MEDIA_ROOT = '/home/usuario/archivos/'` i `upload_to = 'imagenes'` un suposat arxiu *abc.jpg* es guardaria al directori `/home/usuario/archivos/imagenes/abc.jpg`. Si pel contrari s'utilitza `upload_to = '/imagenes'` l'arxiu s'intentaria guardar com `/imagenes/abc.jpg`.

Un inconvenient, més què un problema, ha estat el fet que desconeixia totalment el món de *Python* i *Django*. Dic inconvenient i no problema ja que ha estat un repte constant trobar solució a tots els problemes que han anat sorgint i conèixer aquests dos nous mons.

A l'hora de programar amb *Django* (*Python*) s'ha de tenir molta cura ja que és un llenguatge molt sensible a errors. Un dels errors que es va cometre, i fins gairebé dos dies després no s'hi va trobar la solució, va ser en les definicions dels *models* (*model.py*). En acabar les classes es fan *def __unicode__(self)*. S'ha de ficar dues barres baixes per costat i nosaltres només en ficàvem una. Amb una, el programa funciona però no extreu el resultat adient.

8.- SEGURETAT I CONTROL D'ERRORS

En aquest apartat de seguretat i control d'errors es detallaran les mesures aplicades per evitar enviaments massius o les possibles errades d'usuari, quines s'han tingut en compte i com s'han solucionat. També es comentarà el sistema de còpies de seguretat per evitar problemes de pèrdues d'informació.

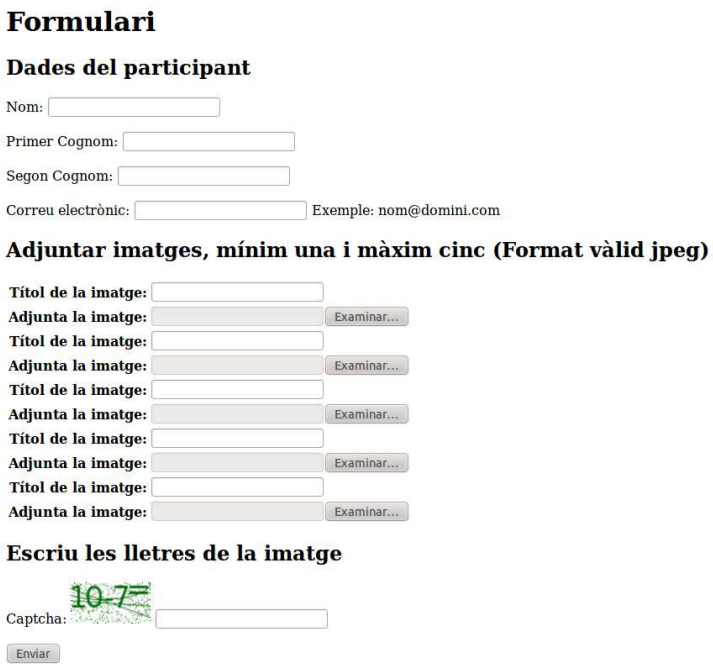
8.1.- SEGURETAT I CONTROL D'ERRORS DESENVOLUPATS

La seguretat i el control d'errors és un tema important a tractar en qualsevol entorn web, el nostre cas no ha estat una excepció.

Anem a exemplificar algunes mesures de seguretat i de control d'errors implementades en el nostre entorn web.

Implementar un bon control d'errors, en la majoria dels casos, és també aplicar una bona mesura de seguretat, ja que a part d'evitar els errors dels usuaris no intencionats, també podem evitar usos indeguts.

A continuació es mostra una imatge del formulari de dades utilitzat en la màquina de proves.



Imatge 82: Formulari ha omplir pel concursant

En aquest formulari s'han implementat diferents mesures de control i al mateix temps de seguretat i control d'errors.

Com l'usuari o participant al concurs ha de complimentar degudament tots els camps del formulari tots ells disposen d'exemples d'introducció. Així, *Nom, Primer Cognom, Segon Cognom i Correu* han de ser omplerts amb valors vàlids.

- Nom espera una entrada de fins a 32 caràcters.
- En el cas dels dos cognoms s'ha ampliat fins a 64 caràcters cadascun.
- El camp correu ha de ser omplert amb una adreça de correu electrònic amb el format nom@domini.com.

Si no s'introdueix algun d'aquests camps el formulari no podrà ser enviat i s'informarà a l'usuari de com ha de fer-ho.

A part de les mesures de control anterior, en tractar-se d'un concurs de fotografia, es va trobar obvi que el concursant hagués d'introduir obligatòriament almenys una imatge amb el seu títol. De no ser així, el sistema tampoc deixarà enviar el formulari.

Per finalitzar amb el control d'errors i implementar una mesura de seguretat extra el participant ha de realitzar l'operació del *Captcha*. Si no es resol correctament tampoc deixarà enviar el formulari.

El *Captcha* és una mesura de seguretat implementada per evitar enviaments massius. Un atac que es pot realitzar utilitzant aplicacions que es dediquen a enviar formularis. Amb això hi ha el perill de bloquejar el servidor i que l'administrador hagi de destriar entre els formularis correctes i els fraudulents.

En el nostre cas es va escollir un *Captcha* numèric ja que el contingut de la web era relacionat amb les matemàtiques.

Imatge 83: Captcha matemàtic a omplir pel concursant

Anem a detallar el control d'errors implementat en el *formulari* i veure com funciona.

Control d'errors en el *formulari*:

La implementació del control d'errors va lligada al *model* pel que fa als camps obligatoris com nom, cognoms i adreça de correu, més el que hem implementat en el formulari directament.

Si el participant només escriu el nom o el primer o segon cognom o es deixa algun dels camps sense emplenar apareix l'error de "*Aquest camp és obligatori*".

Error de camps obligatoris:

Imatge 84: Error en camp obligatori del formulari web

En canvi si el participant emplena bé els tres primers camps però no escriu una adreça de correu vàlida, apareix l'error de li diu "*Introduïu una adreça de correu vàlida*".

Error en el format de l'adreça de correu:

Imatge 85: Error en Correu del formulari web

Si el registre de les dades del participant han estat emplenades correctament però pel contrari no ha adjuntat cap imatge, apareix l'error que diu "*Heu d'enviar almenys una fotografia*".

Error al adjuntar imatge:

Imatge 86: Error adjuntar imatge en el formulari web

Les imatges enviades han de tenir una mida determinada per no saturar els enviaments i que fer que sigui un procés fluid. Per aquest motiu si l'usuari envia una imatge massa pesada apareix el missatge "*La imatge ha de ser menor que 3145728 bytes*".

Error en el pes de la imatge:

Imatge 87: Error en el Pes de la Imatge

Les imatges han de tenir un format específic, *jpeg*. Si el participant envia una imatge en un format diferent, apareix el missatge "*La imatge ha d'estar en format jpeg*".

Error format de la imatge:

Imatge 88: Error format de la Imatge

Com a últim punt del control d'errors del formulari, tenim el *Captcha* per evitar enviaments massius mitjançant *boots*. Consisteix en que el participant ha d'introduir el valor de l'operació matemàtica dins el requadre. Si deixa l'espai en blanc apareix el missatge de "*Aquest camp és obligatori*".

Imatge 89: Error Captcha

Si no realitza l'operació correctament aleshores apareix un error en el *Captcha* que li diu "*Invàlid CAPTCHA*" i canvia el valor de l'operació per a que es torni a introduir.

Imatge 90: Invàlid Captcha

L'elecció del servidor Apache en si representa una altra mesura de seguretat i control, Apache disposa de diferents mecanismes per augmentar la seguretat del lloc web com poden ser filtres i modificar o crear un fitxer de configuracions.

En l'annex 13.2 es pot observar el fitxer de configuració d'Apache on establim un seguit de normes per millorar la seguretat del lloc permetent o denegant accessos i de cara al programador implementant una mesura de control d'errors mol útil com són els *logs d'Apache.*

Aquesta mesura de control s'ha afegit a l'arxiu de configuració *d'Apache* i permet veure quins errors es donen a l'hora d'executar l'aplicació.

```
ErrorLog /var/log/apache2/fotomath-error.log
LogLevel warn
CustomLog /var/log/apache2/fotomath-access.log combined
```

Imatge 91: Errors Log Apache

Les línies que es mostren en la imatge permeten mantenir un registre del que passa al servidor mentre s'executa l'aplicació i poder consultar-ho per corregir o solucionar els problemes que sorgeixen.

El registre d'errors del servidor (*ErrorLog*) és el més important de tots els registres. *Apache* envia qualsevol informació de diagnòstic i registrarà qualsevol error que es doni al processar peticions a l'arxiu de registre seleccionat. És el primer lloc on s'ha de mirar si sorgeix algun problema en iniciar el servidor o durant el seu funcionament normal, ja que amb freqüència es trobarà en ell la informació detallada del que ha anat malament i com solucionar el problema.

Per finalitzar amb les mesures seguretat i control comentar que s'ha implementat un sistema de copies de seguretat de la base de dades.

Tota aplicació informàtica que treballa amb dades o guarda dades (d'usuaris, material, imatges...) necessita mantenir un sistema de còpies de seguretat, si pot ser, que realitzi les còpies, automàticament, de tot el material del que disposa el sistema. En el nostre cas vam implementar un sistema que ens feia una còpia de tota la base de dades.

```
#!/bin/bash

## BEGIN CONFIG ##
HOST=localhost
BACKUP_DIR=tmp
## END CONFIG ##

if [ ! -d $BACKUP_DIR ]; then
  mkdir -p $BACKUP_DIR
fi

POSTGRE_DBS=$(psql -h $HOST -U postgres -l | awk ' (NR > 2) && (/[a-zA-Z0-9]+[ ]+[|]/) && ( $0 !~ /template[0-9]/) { print $1 }');

for DB in $POSTGRE_DBS ; do
  echo "* Backuping PostgreSQL data from $DB@$HOST..."
  pg_dump -h $HOST -U postgres $DB > $BACKUP_DIR/pg_$DB.sql
done
```

Imatge 92: Còpies de seguretat de la Base de Dades

9.- RESULTATS

En aquest apartat es comentaran els resultats obtinguts després de l'elaboració del projecte.

9.1.- RESULTATS OBTINGUTS

El resultat obtingut després de l'elaboració i la implantació del programari desenvolupat, ha estat una web totalment operativa i funcional. Una web amb un sistema de bases de dades al darrera com a suport, que permet gestionar els participants i les imatges que aquests fan arribar. Aquest nou sistema proporciona un procediment molt més àgil que el de l'edició anterior.

En desenvolupar també un sistema d'administració per a la Web li permet al *web màster* realitzar canvis o consultes d'una manera fàcil i senzilla, inclús realitzar modificacions del contingut, dels participants o de les imatges.

A part, s'ha donat al jurat una eina amb la qual poder fer les seves valoracions. En l'anterior edició s'havien de realitzar-les en un full d'Excel amb totes les dades dels participants i de les seves imatges. En la nostra versió, a mesura que el jurat veu les imatges ja es poden valorar.

El formulari del participant s'ha realitzat per tal que fos intuïtiu, poc susceptible a errors i robust en quant a possibles fallades.

En definitiva, crec que s'han assolit els objectius marcats en un principi, d'agilitzar i millorar la gestió dels participants al concurs i les seves imatges, aportant una eina que facilita la tasca corresponent a cada usuari (programador, jurat i participant).

9.2.- ESTUDI DE VIABILITAT ECONÒMICA I LEGAL DEL PROJECTE

Per realitzar l'estudi de la viabilitat econòmica i legal de projecte s'ha d'avaluar si els costos de desenvolupament i de manteniment més els costos d'allotjaments són suportables o el benefici que genera la web es superior a les despeses. A més, s'ha de tenir en compte la legalitat i la normativa.

FotoMath és una *web* operativa que no s'ha desenvolupat per generar cap tipus de benefici material. Per tant, el que s'ha de fet és que tingui unes despeses de desenvolupament i de manteniment mínimes.

Si *FotoMath* fos una web amb despeses s'hagués hagut de buscar alternatives per generar algun benefici. Una de les solucions hagués pogut ser incloure "*banners*" publicitaris, acció molt comú en moltes webs actuals, però que no ha estat necessari en el nostre cas.

El cost del desenvolupament d'aquesta web ha estat zero pel que fa a diners, però el temps que s'ha invertit en el seu desenvolupament si que ha estat important. També s'ha de dir que si la web hagués estat realitzada per un professional de Django i programador de Python aquest temps s'hagués reduït a ¼ part segurament ja que una persona que s'hi dedica professionalment pot fer la mateixa feina en un espai de temps molt més reduït i no necessita invertir temps en la resolució de dubtes com l'hem hagut d'invertir nosaltres.

En estar allotjada en un servidor virtual de la *UdL* la nostra web té un cost d'allotjament zero. No obstant, avui en dia hi ha molts llocs amb allotjament gratuït a canvi d'algun tipus de publicitat. En el cas que la web no pogués continuar essent allotjada pels servidors de la Universitat es podria penjar en algun d'aquests servidors. En el cas que no interesses el fet de tenir publicitat de tercers en l'actualitat existeixen allotjaments per menys de 50€ anuals.

Pel que fa a la protecció de dades dels usuaris, *FotoMath* compleix amb la llei Orgànica (LOPD), que ve a dir:

"La Llei Orgànica 15/1999 de 13 de desembre de protecció de dades de caràcter personal, (LOPD), és una Llei Orgànica espanyola que té per objecte garantir i protegir, pel que fa al tractament de les dades personals, les llibertats públiques i els drets fonamentals de les persones físiques, i especialment del seu honor, intimitat i privacitat personal i familiar."

"El seu objectiu principal és regular el tractament de les dades i fitxers, de caràcter personal, independentment del suport en el qual siguin tractats, els drets dels ciutadans sobre ells i les obligacions d'aquells que els creen o tracten."

En l'article 18.4 es disposa:

"La Llei limitarà l'ús de la informàtica per garantir l'honor i la intimitat personal i familiar dels ciutadans i el ple exercici dels seus drets"

Per aquests fets es pot concloure que *FotoMath* es una web viable, econòmica i que es troba dins de la legalitat.

10.- MILLORES

Tenint en compte que el nostre projecte tenia com a data límit el termini del concurs de la II edició hi va haver coses que no és van poder fer.

En el formulari

Una de les millores a tenir en compte per al proper concurs és que el jurat pugui veure les imatges des de la part administrativa. En l'actualitat només pot votar-les. La intenció era que es mostrés una miniatura de la imatge i un camp al costat per poder-li posar la nota.

En la gestió d'imatges

Seria interessant aconseguir que aquesta aplicació fos escrita en *Django* (*Python*) i no en *JavaScript*. Aquesta millora o canvi serà difícil de dur a terme fins que l'equip de *Django* implementi una bona llibreria.

Com en tot programari lliure, en *Django* van apareixent millores constantment. Pel que s'ha observat en la documentació oficial és una de les millores que hi ha pendents és que la gestió de les imatges sigui més àgil i ràpida del que ho és ara.

11.- CONCLUSIONS

Els organitzadors del certamen *FotoMath* tenien unes necessitats de gestió i organització dels participants al concurs de fotografia. Aquest problema els generava una enorme despesa de temps i esforç.

L'objectiu del projecte ha estat donar una solució a aquests problemes.

Aquet projecte ha servit per poder donar una solució als problemes que tenien, i marcar les bases per a que es pugui continuar utilitzant Django en futures edicions, aquesta és la "grandesa" de Django, un cop s'entén com funciona el *framework* i es coneix el vocabulari i les funcions que aporta l'entorn, els portals i les aplicacions web surten de forma ràpida, sent perfectament mantenibles. Per això, després d'haver treballat amb altres llenguatges de programació *web* recomanaria la utilització de *web frameworks* i més concretament, *Django*.

Amb caràcter personal, considero que he assolit els objectius i coneixements necessaris per donar una solució a un problema real, la presa de decisions, la interacció amb el client i l'usuari, tenir clares les limitacions personals i buscar-hi solucions sent autodidacta, tenir clars els sistemes i tecnologies actuals per adaptar-los a les nostres necessitats i així poder tenir en compte les despeses, i tenir clares les lleis de protecció de dades per gestionar les dels usuaris.

És una satisfacció personal poder donar una solució a un problema i així facilitar la tasca desinteressada dels membres del concurs de fotografia FotoMath, per a properes edicions queda pendent la millora de la part administrativa, fent que el jurat pugui votar directament la imatge que visualitzen.

12.- WEBGRAFIA

En aquest apartat és presentarà la webgrafia que s'ha utilitzat per dur a terme el projecte. En ser *Django* un entorn de treball molt recent no s'ha pogut afegir bibliografia en paper.

S'ha de tenir en compte que en moltes de les webs que segueixen s'hi ha fet varies cerques.

http://es.wikipedia.org/

http://ca.wikipedia.org

http://en.wikipedia.org

http://chuidiang.blogspot.com

http://heim.ifi.uio.no/~trygver/

http://www.djangobook.com/en/2.0/

http://humitos.homelinux.net/django-book-sphinx/

http://django.es/

https://www.djangoproject.com/

http://djangopackages.com/

https://groups.google.com/group/django-es?hl=es&pli=1

http://www.python.org/

http://httpd.apache.org/

http://initd.org

13.- ANNEX I - (CONFIGURACIÓ SERVIDORS)

13.1.- DJANGO.WSGI

```
import os
import sys

os.environ['DJANGO_SETTINGS_MODULE'] = 'settings'

path = '/var/www/fotomath/scripts'
if path not in sys.path:
    sys.path.append(path)

import django.core.handlers.wsgi
application = django.core.handlers.wsgi.WSGIHandler()
```

13.2.- APACHE2

```
<VirtualHost *:80>
ServerName nemessis
DocumentRoot /var/www/fotomath
Alias /sitemedia /var/www/fotomath/sitemedia
Alias /2009 /var/www/fotomath/2009
<Directory /var/www/fotomath/sitemedia>
Order deny,allow
Allow from all
</Directory>
<Directory /var/www/fotomath/2009>
Order deny,allow
Allow from all
</Directory>
Alias /adminmedia /usr/share/pyshared/django/contrib/admin/media
<Directory /usr/share/pyshared/django/contrib/admin/media>
Order allow,deny
Allow from all
</Directory>
<IfModule mod_wsgi.c>
WSGIScriptAlias / /var/www/fotomath/scripts/django.wsgi
</IfModule>
<Directory /var/www/fotomath/scripts>
Order allow,deny
Allow from all
</Directory>
ErrorLog /var/log/apache2/fotomath-error.log
LogLevel warn
CustomLog /var/log/apache2/fotomath-access.log combined
</VirtualHost>
```

14.- ANNEX II - (CODI SERVIDOR DE PROVES)

14.1.- SETTINGS.PY

```
# -*- coding: utf-8 -*-
# Django settings for scripts project.

DEBUG = True
TEMPLATE_DEBUG = DEBUG

ADMINS = (
('nemessis', 'cbosch81@gmail.com'),
)

MANAGERS = ADMINS
DATABASES = {
'default': {
'ENGINE': 'django.db.backends.postgresql_psycopg2',
'NAME': 'projecte_jbd',
'USER': 'usuari_jbd',
'PASSWORD': '123456',
'HOST': 'localhost',
'PORT': '5433',
    }
}

# Local time zone for this installation. Choices can be found here:
# http://en.wikipedia.org/wiki/List_of_tz_zones_by_name
# although not all choices may be available on all operating systems.
# If running in a Windows environment this must be set to the same as
your
# system time zone.
TIME_ZONE = 'Europe/Andorra'

# Language code for this installation. All choices can be found here:
# http://www.i18nguy.com/unicode/language-identifiers.html
LANGUAGE_CODE = 'ca'
SITE_ID = 1
# If you set this to False, Django will make some optimizations so as
not
# to load the internationalization machinery.
USE_I18N = True
# If you set this to False, Django will not format dates, numbers and
# calendars according to the current locale
USE_L10N = True

# Absolute path to the directory that holds media.
# Example: "/home/media/media.lawrence.com/"
```

```
MEDIA_ROOT = ‘/var/www/fotomath/sitemedia/’

# URL that handles the media served from MEDIA_ROOT. Make sure to use a
# trailing slash if there is a path component (optional in other cases).
# Examples: “http://media.lawrence.com”, “http://example.com/media/”
MEDIA_URL = ‘http://nemessis/sitemedia/’
#STATIC_URL=’/static/’
# URL prefix for admin media—CSS, JavaScript and images. Make sure to
use a
# trailing slash.
# Examples: “http://foo.com/media/”, “/media/”.
ADMIN_MEDIA_PREFIX = ‘/adminmedia/’
# Make this unique, and don’t share it with anybody.
#SECRET_KEY = ‘6#_u#^_ks&-6qhgtww8p5zlyz8n6to6sqn#pjbxs(ko^vjb)8j’
# List of callables that know how to import templates from various
sources.
TEMPLATE_LOADERS = (
‘django.template.loaders.filesystem.load_template_source’,
‘django.template.loaders.app_directories.load_template_source’,
#    ‘django.template.loaders.filesystem.Loader’,
#    ‘django.template.loaders.app_directories.Loader’,
#     ‘django.template.loaders.eggs.load_template_source’,
)

MIDDLEWARE_CLASSES = (
‘django.middleware.common.CommonMiddleware’,
‘django.contrib.sessions.middleware.SessionMiddleware’,
‘django.contrib.auth.middleware.AuthenticationMiddleware’,
)

ROOT_URLCONF = ‘urls’
TEMPLATE_DIRS = (
‘/var/www/fotomath/scripts/templates’,
# Put strings here, like “/home/html/django_templates” or
“C:/www/django/templates”.
# Always use forward slashes, even on Windows.
# Don’t forget to use absolute paths, not relative paths.
)

INSTALLED_APPS = (
‘django.contrib.auth’,
‘django.contrib.contenttypes’,
‘django.contrib.sessions’,
‘django.contrib.sites’,
‘django.contrib.admin’,
‘captcha’,
‘participants’,
)
```

14.2.- MODELS.PY

```
# -*- coding: utf-8 -*-
from django.db import models
import datetime


class Concursant(models.Model):
# Creem els caps de concursant per la bd
nom = models.CharField(max_length=32, verbose_name='Nom')
cognom1 = models.CharField(max_length=64, verbose_name='Primer Cognom')
cognom2 = models.CharField(max_length=64, verbose_name='Segon Cognom')
correu = models.EmailField(verbose_name=u'Correu electrònic',
help_text=('Exemple: nom@domini.com'))
def __unicode__(self):
"Retorna el nom complert del concursant"
return u'%s %s %s' % (self.nom, self.cognom1, self.cognom2)
class Meta:
ordering = ['cognom1']

class Imatge(models.Model):
# Creem els caps de imatge per la bd
concursant = models.ForeignKey(Concursant)
nom_imatge = models.CharField(max_length=128, verbose_name=u'Títol de la
imatge')
imatge = models.ImageField(upload_to='images/photos/2011',
verbose_name='Adjunta la imatge')
creada = models.DateTimeField(auto_now_add=True)
modificada = models.DateTimeField(auto_now=True)
def __unicode__(self):
"Retorna el nom de la imatge"
return u'%s' % (self.nom_imatge)
class Meta:
ordering = ['nom_imatge']

class Jurat(models.Model):
# Creem els caps de jurat per la bd
imatge = models.ManyToManyField(Imatge, through='Puntsimatge')
nom = models.CharField(max_length=128, verbose_name='Nom de la persona
del jurat',
help_text=(u'Escriu el teu nom com a jurat'))
def __unicode__(self):
"Retorna el nom de la persona del jurat"
return u'%s' % (self.nom)
class Meta:
ordering = ['nom']

class Puntsimatge(models.Model):
# Creem els caps de puntsimatge per la bd
imatge = models.ForeignKey(Imatge)
jurat = models.ForeignKey(Jurat)
```

```
punts = models.IntegerField(max_length=2, verbose_name=’Puntuació de la
imatge’,
help_text=(u’Enter 2 dígits sense decimals’))
def __unicode__(self):
“Retorna les puntuacions de les imatges”
return u’%s’ % (self.punts)
class Meta:
ordering = [’punts’]
```

14.3.- ADMIN.PY

```
# -*- coding: utf-8 -*-
from django.contrib import admin
from participants.models import Concursant, Imatge, Jurat, Puntsimatge


class PuntsimatgeInline(admin.TabularInline):
# Definim la class puntsimatge per la part administrativa
model = Puntsimatge
extra = 1

admin.site.register(Puntsimatge)

class ImatgeInline(admin.TabularInline):
# Definim la class imatge i li donem atributs per a poder adjuntar fins
a 5 imatges
model = Imatge
max_num = 5
extra = 1


class AdminConcursant(admin.ModelAdmin):
# Definim els caps a mostrar en la part administrativa pel que fa al
concursant
list_display = (‘nom’, ‘cognom1’, ‘cognom2’, ‘correu’)
list_filter    = (‘cognom1’,)
ordering = (‘cognom1’,)
search_fields  = (‘nom’, ‘cognom1’, ‘cognom2’,)
inlines = [ImatgeInline,]

admin.site.register(Concursant, AdminConcursant)

class AdminImatges(admin.ModelAdmin):
# Definim els caps a mostrar en la part administrativa pel que fa a la
imatge
fieldsets = (
(None, {’fields’: (‘nom_imatge’, ‘imatge’)}),
        )
list_display = (‘concursant’, ‘nom_imatge’, ‘imatge’, ‘creada’,
‘modificada’,)
list_filter    = (‘nom_imatge’,)
search_fields  = (‘nom_imatge’,)

admin.site.register(Imatge, AdminImatges)

class AdminJurat(admin.ModelAdmin):
# Definim els caps a mostrar en la part administrativa pel que fa al
jurat
list_display = (‘nom’,)
search_fields  = (‘nom’,)
```

```
inlines = [PuntsimatgeInline,]

admin.site.register(Jurat, AdminJurat)
```

14.4.- FORMS.PY

```
# -*- coding: utf-8 -*-
from django import forms
from participants.models import Concursant, Imatge
from captcha.fields import CaptchaField
from django.forms.models import BaseInlineFormSet, inlineformset_factory


class ConcursantForm(forms.ModelForm):
# Definim el formalari concursant
class Meta:
model = Concursant
fields = (‘nom’, ‘cognom1’, ‘cognom2’, ‘correu’)

class ImatgeBaseInlineFormSet(BaseInlineFormSet):
# Definim el formulari imatge base per poder fer validacions dels camps
def clean(self):
“”” Obté els formularis que tenen dades correctes”””
super(ImatgeBaseInlineFormSet, self).clean()
contador = 0
for form in self.forms:
try:
if form.cleaned_data:
contador += 1
except AttributeError:
# Si un dels subformularis no és correcte,
# informa de AttributeError per al cleaned_data
pass
if contador < 1:
raise forms.ValidationError(“Heu d’enviar almenys una fotografia”)

class ImatgeForm(forms.ModelForm):
# Definim el formulari imatge
class Meta:
model = Imatge
fields = (‘nom_imatge’, ‘imatge’)

def clean_imatge(self):
“”” Rebutja les imatges > 3MB i les que no estan en format jpeg “””
max_mida = 3*2**20
formats_imatge = [’image/jpeg’]
imatge = self.cleaned_data[’imatge’]
tipus_imatge = imatge.content_type
if tipus_imatge in formats_imatge:
if imatge._size > max_mida:
raise forms.ValidationError(
‘La imatge ha de ser menor que %d bytes.’ % max_mida
                    )
else:
raise forms.ValidationError(“La imatge ha d’estar en format jpeg”)
```

```
return imatge

ImatgeFormSet = inlineformset_factory(
Concursant,
Imatge,
form = ImatgeForm,
formset = ImatgeBaseInlineFormSet,
can_delete=False,
extra=5)


class CaptchaTestModelForm(forms.ModelForm):
# Definim el formulari captcha per poder fer validacions del camp
captcha = CaptchaField()
class Meta:
model = Concursant
        exclude = (‘nom’, ‘cognom1’, ‘cognom2’, ‘correu’,)
```

14.5.- VIEWS.PY

```
# -*- coding: utf-8 -*-
from django import forms
from django.template import TemplateDoesNotExist, RequestContext
from captcha.fields import CaptchaField
from django.shortcuts import get_object_or_404, render_to_response
from django.http import HttpResponse, Http404, HttpResponseRedirect
from django.conf import settings
from participants.forms import ConcursantForm, ImatgeFormSet,
CaptchaTestModelForm
from participants.models import Concursant, Imatge
import unicodedata
from django.core.urlresolvers import reverse


def gestio_concursants(request):
# Si el formulari ha estat enviat:
if request.method == 'POST':
# El formulari del concursant emplenat:
concursant_form = ConcursantForm(request.POST)
captcha_form = CaptchaTestModelForm(request.POST)
# Validació del formulari concursant:
if concursant_form.is_valid():
concursant = concursant_form.save(commit=False)
imatge_formset = ImatgeFormSet(request.POST, request.FILES,
instance=concursant)
# Validació del formulari foto i del capcha:
if imatge_formset.is_valid() and captcha_form.is_valid():
concursant.save()
imatge_formset.save()
captcha_form.save()
human = True

return HttpResponseRedirect('/gracies/')
else:
imatge_formset = ImatgeFormSet(instance=Concursant())
else:
concursant_form = ConcursantForm()
imatge_formset = ImatgeFormSet(instance=Concursant())
captcha_form = CaptchaTestModelForm()
return render_to_response('formulari.html', {
'MEDIA_URL': settings.MEDIA_URL,
'concursant_form': concursant_form,
'imatge_formset': imatge_formset,
'captcha_form': captcha_form},
context_instance=RequestContext(request))

def gracies(request):
pag = "gracies.html"
return render_to_response(pag)
```

14.6.- URL.PY

```
# -*- coding: utf-8 -*-
from django.conf.urls.defaults import *
from views import gestio_concursants, gracies
from django.contrib import admin
from settings import MEDIA_ROOT
admin.autodiscover()

urlpatterns = patterns(“,
url(r’^captcha/’, include(‘captcha.urls’)),
(r’^formulari/’, gestio_concursants),
(r’^gracies/’, gracies),
(r’^admin/’, include(admin.site.urls)),
)
```

14.7.- FORMULARI.HTML

```
<html>
<title>Formulari entrada dades</title>
<body>
<h1>Formulari</h1>
<h2>Dades del participant</h2>
<form enctype='multipart/form-data'  action='.' method='post'>{%
csrf_token %}
{{ concursant_form.as_p }}
<table>
<h2>Adjuntar imatges, mínim una i màxim cinc (Format vàlid jpeg)</h2>
{{ imatge_formset.management_form }}
{{ imatge_formset.non_form_errors }}
{{ imatge_formset.as_table }}
</table>
<h2>Escriu les lletres de la imatge</h2>
{{ captcha_form.as_p }}

<p><input type="submit" value="Enviar" /></p>
</form>
</body>
</html>
```

15.- ANNEX III - (CODI SERVIDOR FOTOMATH)

Arxius modificats o nous per al servidor.

15.1.- VIEWS.PY

```
# -*- coding: utf-8 -*-
from django.http import Http404, HttpResponseRedirect
from django.template import TemplateDoesNotExist, RequestContext
from django.shortcuts import render_to_response
from django.conf import settings
from siteapp.models import Concursant, Imatge
from siteapp.forms import ConcursantForm, ImatgeFormSet, CaptchaTestForm
import os
import Image

def gestionar_concursant(request):
formulari_correcte = False
imatges_concursant = “
# Si el formulari ha estat enviat:
if request.method == ‘POST’:
# El formulari del concursant emplenat:
formulari_concursant = ConcursantForm(request.POST)
formulari_captcha = CaptchaTestForm(request.POST)
# Validació del formulari concursant:
if formulari_concursant.is_valid():
concursant = formulari_concursant.save(commit=False)
try:
concursant.pk = Concursant.objects.get(correu=concursant.correu).pk
except Concursant.DoesNotExist:
pass
formulari_imatge = ImatgeFormSet(request.POST, request.FILES,
instance=concursant)
if formulari_imatge.is_valid() and formulari_captcha.is_valid():
formulari_correcte = True
concursant.save()
formulari_imatge.save()
imatges_concursant = Imatge.objects.filter(concursant=concursant)

else:
formulari_imatge = ImatgeFormSet(request.POST, request.FILES,
instance=Concursant())
else:
formulari_concursant = ConcursantForm()
formulari_captcha = CaptchaTestForm()
formulari_imatge = ImatgeFormSet(instance=Concursant())
return render_to_response(‘formulari.html’, {
```

```
‘MEDIA_URL’: settings.MEDIA_URL,
‘formulari_concursant’: formulari_concursant,
‘formulari_imatge’: formulari_imatge,
‘formulari_captcha’: formulari_captcha,
‘formulari_correcte’: formulari_correcte,
‘imatges_concursant’: imatges_concursant,
}, context_instance=RequestContext(request))
def mostrar_imatges(request):
total_imatges = Imatge.objects.all()
if len(total_imatges) >0:
imatge0 = total_imatges[0]
nom_directori_diapositives = ‘.diapositives’
        #
url_imatges = os.path.split(imatge0.imatge.url)[0]
url_diapositives = os.path.join(url_imatges, nom_directori_diapositives)
        #
directori_imatges = os.path.split(imatge0.imatge.path)[0]
directori_diapositives = os.path.join(directori_imatges,
nom_directori_diapositives)
try:
os.mkdir(directori_diapositives, 0755)
except os.error, msg:
pass
        #
llista_diapositives = [f for f in os.listdir(directori_diapositives)
if os.path.isfile(os.path.join(directori_diapositives, f))]
        #
dades_imatges = []
for f in total_imatges:
if os.path.isfile(f.imatge.path):
nom_diapositiva = ‘dp_’ + os.path.split(f.imatge.path)[1]
diapositiva = os.path.join(url_diapositives, nom_diapositiva)
                #
if nom_diapositiva not in llista_diapositives:
fer_diapositiva(f.imatge)
                #
foto = f.imatge.url
titol = f.titol
autor = “ #
dades = {’autor’:autor, ‘foto’:foto, ‘titol’:titol,
‘diapositiva’:diapositiva}
dades_imatges.append(dades)
return render_to_response(‘expo.html’, {
‘MEDIA_URL’: settings.MEDIA_URL,
‘dades_imatges’: dades_imatges,
}, context_instance=RequestContext(request))

def fer_diapositiva(fitxer, mida=’104x104’):
f = fitxer.path
x, y = [int(x) for x in mida.split(‘x’)]
    #
```

```
cami_a_fitxer, nom_fitxer = os.path.split(f)
nom_diapositiva = ‘dp_’ + nom_fitxer
diapositiva = os.path.join(cami_a_fitxer, ‘.diapositives’,
nom_diapositiva)
    #
imatge = Image.open(f)
imatge.thumbnail([x, y], Image.ANTIALIAS)
    #
try:
imatge.save(diapositiva, imatge.format, quality=90, optimize=1)
except:
imatge.save(diapositiva, imatge.format, quality=90)
```

15.2.- URLS.PY

```
# -*- coding: utf-8 -*-
from django.conf.urls.defaults import *
# from django.views.generic.simple import direct_to_template
from views import static_pages, the_page
from siteapp.views import gestionar_concursant, mostrar_imatges

# Uncomment the next two lines to enable the admin:
from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns(“,
(r’^$’, the_page, {’pagename’: ‘inici’}),
    #
# (‘^2009/(|index.html)$’,
#  direct_to_template, {’template’: ‘2009/index.html’}
    # ),
# (‘^2009/html/(\w+).html$’, static_pages),
    #
(r’^(inici|bases|premis|entrega|jurat|lligams)$’, the_page),
(r’^formulari$’, gestionar_concursant),
(r’^expo$’, mostrar_imatges),
    #
url(r’^captcha/’, include(‘captcha.urls’)),
    #
(r’^admin/’, include(admin.site.urls)),
)
```

15.3.- EXPO.HTML

```
{% extends “plantilla.html” %}
{% block headexpo %}
<script type=”text/javascript” src=”{{ MEDIA_URL }}js/highslide-
full.js”></script>
<link rel=”stylesheet” type=”text/css” href=”{{ MEDIA_URL
}}css/highslide.css” />
<!--[if lt IE 7]>
<link rel=”stylesheet” type=”text/css” href=”{{ MEDIA_URL
}}css/highslide-ie6.css” />
<![endif]-->
<script type=”text/javascript”>
hs.cssDirection = ‘ltr’;
hs.loadingText = ‘Carregant...’;
hs.loadingTitle = ‘Feu clic per cancel·lar’;
hs.focusTitle = ‘Feu clic per portar al capdavant’;
hs.fullExpandTitle = ‘Ampliar a mida real (f)’;
hs.creditsText = ‘Desenvolupat per <i>Highslide JS</i>’;
hs.creditsTitle = ‘Anar a la pàgina principal de Highslide JS’;
hs.previousText = ‘Anterior’;
hs.nextText = ‘Següent’;
hs.moveText = ‘Moure’;
hs.closeText = ‘Tancar’;
hs.closeTitle = ‘Tancar (esc)’;
hs.resizeTitle = ‘Canviar la mida’;
hs.playText = ‘Reproducció’;
hs.playTitle = ‘Reproducció de diapositives (barra espaiadora)’;
hs.pauseText = ‘Pausa’,
hs.pauseTitle = ‘Pausa de diapositives (barra espaiadora)’;
hs.previousTitle = ‘Anterior (fletxa esquerra)’;
hs.nextTitle = ‘Següent (fletxa dreta)’;
hs.moveTitle = ‘Moure’;
hs.fullExpandText = ‘1:1’;
hs.number = “;
hs.restoreTitle = ‘Feu clic per tancar la imatge, feu clic i arrossegueu
per moure. Utilitzeu les tecles de fletxa per següent i anterior.’;
hs.graphicsDir = ‘{{ MEDIA_URL }}images/highslide/’;
hs.align = ‘center’;
hs.transitions = [’expand’, ‘crossfade’];
hs.fadeInOut = true;
hs.dimmingOpacity = 0.8;
hs.outlineType = ‘rounded-white’;
hs.captionEval = ‘this.thumb.alt’;
hs.marginBottom = 105;
hs.numberPosition = ‘caption’;
hs.showCredits = false;

hs.addSlideshow({
interval: 5000,
```

```
repeat: false,
useControls: true,
overlayOptions: {
className: ‘text-controls’,
position: ‘bottom center’,
relativeTo: ‘viewport’,
offsetY: -60
    },
thumbstrip: {
position: ‘bottom center’,
mode: ‘horizontal’,
relativeTo: ‘viewport’
    }
  });
</script>
{% endblock %}

{% block expo %}current{% endblock %}
{% block contingut %}
<h2>Exposici&oacute;</h2>
<p>Fotograf&iacute;es participants en aquest cert&agrave;men.</p>
<div class=”highslide-gallery” style=”width: 600px; margin: auto”>
{% for dada in dades_imatges %}
<a class=’highslide’ href=’{{ dada.foto }}’ onclick=”return
hs.expand(this)”>
<img src=’{{ dada.diapositiva }}’ alt=’{{ dada.titol }}’/></a>
{% endfor %}
</div>
{% endblock %}
```